A critical review on the implementation of static data sampling techniques to detect network attacks

Suzan Hajj, Rayane El Sibai, Jacques Bou Abdo, Jacques Demerjian, Christophe Guyeux, Abdallah Makhoul, Dominique Ginhac

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

A critical review on the implementation of static data sampling techniques to detect network attacks

SUZAN HAJJ [1], RAYANE EL SIBAI[2], JACQUES BOU ABDO[3], JACQUES DEMERJIAN [4], CHRISTOPHE GUYEUX[5], ABDALLAH MAKHOUL[6], AND DOMINIQUE GINHAC [7]

[1]Université de Bourgogne Franche-Comté, Dijon, France
[2]Computer Science Department, Faculty of Sciences, Al Maaref University, Beirut, Lebanon (e-mail: rayane.elsibai@mu.edu.lb)
[3]University of Nebraska at Kearney, USA (e-mail: bouabdoj@unk.edu)
[4]LaRRIS, Faculty of Sciences, Lebanese University, Fanar, Lebanon (e-mail: jacques.demerjian@ul.edu.lb)
[5]Femto-ST Institute, UMR CNRS 6174, Université de Bourgogne Franche-Comté, Besançon, France (e-mail: christophe.guyeux@univ-fcomte.fr)
[6]Femto-ST Institute, UMR CNRS 6174, Université de Bourgogne Franche-Comté, Besançon, France (e-mail: abdallah.makhoul@univ-fcomte.fr)
[7]Université de Bourgogne Franche-Comté, Dijon, France (e-mail: dominique.ginhac@ubfc.fr)

Corresponding author: Rayane El Sibai (e-mail: rayane.elsibai@mu.edu.lb).

ABSTRACT Given that the Internet traffic speed and volume are growing at a rapid pace, monitoring the network in a real-time manner has introduced several issues in terms of computing and storage capabilities. Fast processing of traffic data and early warnings on the detected attacks are required while maintaining a single pass over the traffic measurements. To palliate these problems, one can reduce the amount of traffic to be processed by using a sampling technique and detect the attacks based on the sampled traffic. Different parameters have an impact on the efficiency of this process, mainly, the applied sampling policy and sampling ratio. In this paper, we investigate the statistical impact of sampling the network traffic and we quantify the amount of deterioration that the sampling process introduces. In this context, an experimental comparison of existing sampling techniques is performed based on their impact on several well-known statistical measures.

INDEX TERMS Data sampling, Data streams, Intrusion Detection System (IDS), Statistical analysis.

I. INTRODUCTION

With the emergence of new technologies and applications, the speed and volume of Internet traffic are increasing very rapidly. Current businesses' needs require developing of advanced information networks integrating various technologies such as distributed storage systems, encryption/decryption mechanisms, remote and wireless access, etc. Consequently, Internet service providers and network managers are encouraged to better understand network behavior through the analysis and monitoring of traffic inside the network. Hence, the need for network-based security systems, such as Intrusion Detection Systems (IDSs) [1].

IDSs play an important role in ensuring network security, as they observe user activity on the network and detect any security violation. Usually, Intrusion Prevention Systems (IPSs) can be used at first to ensure the safety of the network and protect it from attacks. For instance, a firewall can be used to manage access inside a private network. It prevents end-users inside a protected network from sending or receiving messages forbidden by the predefined network security policy, without any capability of detecting anomalies or any specific pattern among the network traffic data [2]. As the network becomes more complex, the more exploitable and vulnerable it will be to attacks. Network attacks can be caused by external intruders attempting to access the network, or from legitimate users trying to misuse their granted permissions and to gain more privileges for which they are not authorized. Such abnormal activities are manifested by higher consumption of network resources and many undesired requests overloading it. For instance, the main objective of the DoS attack is to deny end-users from benefiting from network-provided services [3].

The success of an IDS is challenged by the network's implementation flaws and the complexity of the attacks, where it also has to deal with the availability and heterogeneity of the traffic data sources to find malicious behaviors. IDSs' performance is another challenge, as real-time network monitoring requires very fast processing and inspection of network traffic. The amount of data generated in the network represents thus a major problem. When the traffic is huge, the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3118605, IEEE Access

IDS will be unable to inspect every arriving packet.

As defined by the Institute of Electrical and Electronics Engineers (IEEE), the Internet of Things (IoT) is a collection of sensors that form a network connected to the Internet. IDSs designed for IoT environments should be implemented on programmable devices, such as FPGAs, to facilitate adaptation to IoT environments. Confidentiality, integrity, and availability are three important security concepts for applications and services in intelligent IoT-based environments. The implementation of a robust security mechanism in IoT systems depends on the security strength of IoT devices, which in turn depends on several factors, such as power, memory, hardware, software, choices design, etc. These present security challenges in IoT systems and impact the performance of the IDS [4]. Collaboration between IoT devices and the router to shift the compute load from resource-constrained IoT devices to the resource edge router is necessary to increase network lifespan and reduce intrusion detection time.

Since intrusion detection is the process of monitoring the network and detecting the attacks, the data exchanged over the network and coming from different sources such as cell phones, computers, etc., as well as all network activity, have to be measured and processed by the IDS. This latter detects anomalies and sends security alerts to the network administrator as soon as an attack is detected [5]. However, the extensive applications of high-speed Internet make it impossible to adopt traditional packet measuring and processing technologies of network traffic. In fact, these technologies are not scalable to high-speed networks. The largest obstacles in high-speed networks are the huge volume of traffic data to deal with, and the rate at which information accumulates [6].

Real-time and fast processing of traffic data is required: the analysis time of an IP packet must be shorter than the packets' inter-arrival time while maintaining a single pass over the traffic data. Also, early warnings on the detected attacks and their sources must be triggered, requiring, thus, a highly scalable IDS with architecture, storage, and computing capabilities and resources able to support very high throughput. The reason for the inability of current solutions to detect intrusions in high-speed networks is the high cost of using traditional network monitoring schemes. These schemes measure the network parameters of every packet that passes through the network, making it challenging to monitor the behavior of a large number of users in high-speed networks. To keep track of the huge volume of network traffic, one of the possible solutions is to increase the storage and computing resources of the IDS by distributing network packets to multiple IDSs [7], [8]. However, these solutions are expensive.

Applying traditional Machine Learning (ML) approaches in IoT environments may not be suitable because are of the computing and energy constraints of IoT devices. In fact, ML algorithms have complexity issues such as memory complexity, computation, etc. They also lack scalability and are limited to low-dimensional problems. Therefore, using traditional ML approaches is not suitable in such environments with limited resources. As for intelligent IoT devices, the detection of anomalies and intrusions requires real-time data processing, however, traditional ML approaches are not designed to handle real-time data streams. In addition, ML algorithms assume that the entire data is available for processing during the learning phase, which is not true for IoT data. This poses many challenges when traditional ML approaches have to process a huge amount of data, especially, when the data dimensionality is high [9]. This discussion is applicable for security-related functions in IoT where real-time data is processed to identify anomalies, intrusions, etc. Due to these limitations, it is important to combine ML with streaming solutions, such as sampling algorithms. It is, therefore, necessary to filter this data on the fly and store only those that are relevant by carrying out summaries (samples) before applying ML algorithms.

A. DATA SAMPLING

To address the issues presented above, and help the IDS process the information gathered during the data fusion from the routers, switches, firewalls, etc., network packets may be sampled where the router inspects every n-th packet using a sampling technique, and then, records its associated features [10]. Thus, intrusions will be detected based on the sampled data instead of the entire traffic, as shown in Figure 1. Therefore, the IDS can benefit from the available computing and storage resources to analyze the network traffic. The challenge is to prevent the intrinsic loss of information during the sampling process which will lead to low detection accuracy. Over the years, different sampling strategies were investigated in the literature to improve attack detection accuracy. Researchers proposed a variety of static and dynamic sampling algorithms for network traffic reduction. With static sampling algorithms, traffic measurements are sampled either periodically or randomly at a specific predefined interval or using a specific rule. Using a static sampling algorithm to reduce the network traffic volume reduces the bandwidth and storage requirements, making this type of sampling algorithm very efficient. In their turn, dynamic sampling algorithms, also called adaptive sampling algorithms, used different sampling intervals and/or rules to sample the data. In this paper, we will focus on static sampling algorithms.

Selecting a sample of packets from the entire traffic is a challenging task. For instance, if malicious packets are not selected by the sampling algorithm, the attack may not be detected, as the IDS only analyzes the sampled traffic. Therefore, an efficient sampling algorithm must ensure the sampling of the packets that carry a malicious payload. Previous research studies showed that the sampling process can affect, skew, and also distort the anomaly detection metrics and detection rates [11]–[13]. Therefore, choosing an appropriate sampling algorithm and sampling interval that provides a good representation of the overall and original traffic is very important and delicate in case a sampling process is to be applied.

In recent years, the field of sampling network traffic has

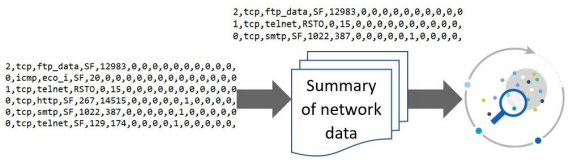


FIGURE 1: Packets sampling of network traffic.

been widely explored by the research community, which has led to the publication of numerous works and benchmarking papers. In this light, several studies examined the impact of data sampling. Mai et al. [13] evaluated in their work the effect of sampling the traffic of high-speed IP-backbone networks on the intrusions' detection results, especially, on the port scans and volume anomaly detection. Different sampling algorithms were used to sample the traffic packets. [14], [15] also assessed the impact of packets sampling on anomalies' detection results. Roudiere et al. [16] evaluated the accuracy of AATAC detector in detecting DDoS attacks over sampled traffic. Different sampling policies were used to sample the traffic. Bartos et al. [17] studied the effect of traffic sampling on anomaly detection, and proposed a new adaptive flow-level sampling algorithm to enhance the accuracy of the sampling process. Silva et al. [18] introduced a framework to evaluate the impact of packets sampling. They discussed the performance of each sampling algorithm and proposed a set of metrics that allows the accurate evaluation of each sampling technique in producing a representative sample of the original traffic. Brauckhoff et al. [11] used traces containing the Blaster worm to assess the accuracy of different anomaly detection and data sampling algorithms.

B. PROBLEM DEFINTION AND MOTIVATION

Nowadays, deploying IDSs inside companies is a common practice to prevent and/or mitigate both internal and external attacks. However, with the restricted bandwidth of network links, and the limited storage and computing resources of the IDSs, it has become difficult to efficiently monitor and manage the network. A possible solution to handle this problem is to apply a sampling strategy to decrease the amount of traffic to be processed. Current research studies are investigating which sampling policy and which parameters provide the best compromise in terms of IDS performance (response time) while having a high attack detection rate.

A sampling policy aims to provide an estimation of a metric of interest from a set of data while reducing the processing cost. This is achieved by selecting a subset of data called "sample" and estimating the metric of interest from this subset. The sampling strategy specifies how the subset of data is selected [19]. More specifically, the packets sampling process aims to construct a sample of data on which future analysis tasks will be carried out. Different parameters affect the efficiency of the sampling and the precision of the estimation of the traffic characteristics, mainly the sampling strategy and the sampling rate. Thus, given the original set of packets, the most difficult task is to select the right sampling policy and the relevant parameters.

Packet sampling is involved in large network monitoring, management, and engineering tasks. It serves to provide a dynamic overview of the network by providing detailed information that can be exploited to infer various estimates, statistics, and aggregates of the traffic. These include the number of packets, the size of packets, the interarrival delays and protocols of packets, the traffic flows, etc. which can be eventually used to detect particular network problems.

As stated by El Sibai et al. [20], A sampling algorithm can be qualified according to the following metrics: (1) Single-pass over the data: since it is almost impossible to store all the traffic packets for further processing, any sampling algorithm must be able to construct the sample by making only one pass over the data. (2) Memory consumption: the sample size affects also the sample quality. The size is usually proportional to the size of the traffic and it depends on the used sampling ratio. The higher the sampling ratio, the higher the accuracy of the sample, but the bigger its size. (3) Skewing ability: a sample must represent the entire network traffic. With the probabilistic sampling methods, all the packets are sampled with the same probability. However, in some cases, some packets should have a higher probability of being sampled (depending on their values, arrival time, etc.). One way to do this is to associate weights to packets and to sample them according to their relative weights. (4) Complexity: due to the high traffic arrival rate, a low complexity is needed to decrease the execution time of sampling. In many applications, packet sampling can be used to provide accurate results while reducing data processing costs. It can be used for controlling network congestion, detecting broken links, misconfigured devices, and rouge network servers. Packet sampling can be also used to verify the quality of service in the network, to build trends, and forecast bandwidth and other resource requirements.

Packets sampling is a class of data sampling techniques that consider packets as basic elements. Therefore, all the packets observed are considered as the original data set and the selected packets are representing the sample. The target sample size is affected usually by the sampling interval, also denoted as the sampling ratio. However, with some sampling algorithms, the obtained sample size can deviate from the target one. The sampling decision of a packet can be of three types: count-based, time-based, and content-based. With the count-based sampling strategies, the sampling decision of a packet is based on its position in the sequence of packets called stream. With the time-based sampling strategies, the sampling decision is based on the packet arrival time. Finally, with the content-based sampling strategies, the sampling decision is based on the packet's content. Content-based sampling strategies are also called filtering algorithms and are outside the scope of this work.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3118605, IEEE Access

C. OBJECTIVE AND CONTRIBUTIONS

The aim of this study is to check whether summarized (sampled) are sufficient to detect attacks in high-speed networks. We aim at quantifying the robustness of the traffic characteristics under different sampling strategies. The sampling process selects some traffic items from the whole received traffic based on the used sampling policy and chosen sample size. Thus, sampling is considered to be an approximate method of measurement. Since in the sampled dataset many packets are not sampled, the traffic distribution of sampled data may deviate from the distribution and statistical characteristics of the original traffic. Taking these sampled data as the input of any attack detection algorithm will inevitably affect the accuracy of the detection algorithm.

An effective sampling policy selects a subset of packets with which the statistical parameters of the traffic can be estimated accurately. In this context, we focus in this work on the statistical impact of packet sampling on traffic analysis. The sampling techniques are evaluated and compared in terms of the similarity between the results of the aggregation query evaluated on the original traffic and the sampled one. By building a sample of the traffic and performing offline analysis of the sampled packets, several statistical metrics gathered from sampled traffic and non-sampled traffic can be measured and compared to assess the level of degradation introduced by the sampling process. This study is initiated with a survey of sampling algorithms. Then, the experimental comparison of all these sampling methods is performed.

Recently published benchmarks, cited in Table 1, focus on the most traditional sampling algorithms. However, none of these works have thoroughly reviewed all data sampling approaches, their impact on detecting a variety of attacks, and the behavior and robustness of the features under different sampling strategies. Even if it is well known that the sampling distorts the statistics measures, it was surprising that few studies have explored how the network characteristics estimation varies according to the used sampling method, sample size, etc. and how this affects statistical inference from these data.

Pescape et al. [14] considered as a list of traffic characteristics. Two statistical metrics to assess feature distortions are used, "Hellinger" for similarities and "Fleiss Chi-Square" for classification. The intersection of the chosen characteristics sets in each database separately formed a robust final feature set. Various sampling techniques are applied to assess the robust feature set. Results showed that sampling techniques have a slight impact on reducing the degradation behavior of the anomaly detection process and that the characteristics of many packets are most affected by distortion. Data sampling depends on the method of measuring the data. Therefore, sampling is subject to a certain variance in the total traffic distribution that affects the accuracy of the anomaly detection results. To deal with this problem, Pan et al. [21] suggested a method of measuring packet sampling based on the IP flow. The sampling rate is variable and depends on the arrival process sequence of the IP flow at both packet and flow levels and the flow size. Subsequently, the sampling probability is adjusted based on the number of unlike samples in the stream. Two evaluation metrics are used, RMSE to measure volume anomalies in the size of the IP stream and the hit detection rate to measure the sequence of variance in the stream. The results showed that for a sample rate of 1%, the proposed solution detected 27 out of 30 of worm and DDoS attacks, while traditional random sampling only detected three. Singh et al. [22] studied the statistical impact of data sampling on traffic analysis by calculating the statistical parameters for unsampled data set and then for sampled data. Subsequently, a comparative analysis of the unsampled and sampled data sets was performed. The following sampling algorithms were used to sample the data: Simple Random Sampling (SRS), Systematic sampling, and under-over sampling. The following attributes from the NSL KDD dataset were considered: duration, src_bytes, dst_bytes, wrong_fragment, num_compromised, num_file_ creations, and srv_count. The statistical parameters used to compare the network characteristics before and after sampling were: the mean, the range including the minimum and maximum values for each attribute, and the standard deviation showing the distribution of the network. Silva et al. [18] developed a data sampling framework based on a multi-layered design. The framework selects the characteristics and sampling technique according to the measurement task. The implementation of the framework is based on a sampling taxonomy that determines the granularity, the selection scheme, and the selection trigger.

A comparison between the above papers and our work is summarized in Table 1.

In this paper, the impact of sampling is studied in terms of well-known statistical metrics, such as the mean, standard deviation, median, etc. from the perspective of determining the characteristics of the traffic before and after sampling. Our main objective is to provide an up-to-date survey of static sampling algorithms and evaluate the data sampling impact on network traffic analysis. Different sampling algorithms and a variety of parameters are considered during our study. Our contributions in this paper can be summarized as follows. (1) Presenting an exhaustive survey of existing data stream sampling algorithms, (2) Elaborating on the following critical question: Given the network traffic, what are the suitable sampling policy and parameters to apply to reduce the network volume? (3) Evaluating the behavior and robustness of various features characterizing the network, under different sampling strategies and parameters, (4) Finding out which attacks are more robust to the sampling process, (5) Finding out whether there exists a family of features which is more robust to the sampling process, and (6) Exposing, based on the obtained results, the research challenges and possible solutions to handle them.

D. SUMMARY AND OUTLINE

The rest of the paper is organized as follows. Section I-B discusses the motivation and introduces the contribution of our work. This section also shows the difference between

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3118605, IEEE Access

TABLE 1: Comparison of this work and similar recent works, where C1 shows if the benchmark evaluated all existing sampling algorithms, C2 represents if the benchmark studied the suitable sampling policy and parameters to apply to reduce the network volume, C3 depicts if the benchmark presented an exhaustive survey of existing data stream sampling algorithms, C4 shows if the benchmark evaluated the impact of sampling on different types of attack, C5 shows if the benchmark studied the impact of sampling on the behavior of different features, C6 shows if the benchmark compared the performance of different sampling techniques with respect to their accuracy, and C7 shows if the benchmark evaluated the statistical measures that assess distortion caused by sampling on traffic with unsampled ones.

Reference	Year	C1	C2	C3	C4	C5	C6	C7
Pescape *et al.* [14]	2010	✓	✓	✗	✗	✓	✓	✓
Pan *et al.* [21]	2012	✓	✓	✗	✓	✗	✗	✓
Singh *et al.* [22]	2014	✓	✓	✗	✗	✓	✓	✓
Silva *et al.* [18]	2015	✓	✓	✗	✗	✗	✓	✓
This paper	2021	✓	✓	✓	✓	✓	✓	✓

our work and existing ones. Section II presents a survey of existing sampling algorithms. Section III elaborates on the experimental approach and methodology that we will apply in our work. Section IV illustrates and discusses the obtained results. Finally, Section V concludes the paper.

II. TAXONOMY OF PACKETS SAMPLING POLICIES

In high-speed networks, network traffic arrives continuously, at a high rate. The IDS receiving this traffic may not be able to store them exhaustively, and/or have sufficient computing resources to process them rapidly. Thus, processing high-speed network traffic requires minimizing the volume of traffic by building, storing, and maintaining a sample of this traffic. An efficient sample should be able to answer, approximately, to any query regardless of the period investigated. The literature shows that most sampling techniques share the following main components: sampling function, the temporal aspect of packets, windowing model, and sample size [19]. Classifying and evaluating sampling policies according to these components is a very important step for reaching better sampling accuracy.

In the following, Figure 2 presents our taxonomy that fragments the sampling policies into the four components: sampling function, the temporal aspect of packets, windowing model, and sample size. Table 2 classifies existing sampling policies according to the proposed taxonomy.

- Sampling function: it identifies the policy defining which packets will be added to the sample. This policy may follow a static approach or a dynamic one. A static approach can be either deterministic or random.
- Temporal aspect of packets: it can be physical or logical (sequential). The physical aspect depicts the arrival time of the packet, the logical aspect describes the index of the packet in the traffic stream.
- Windowing model: sampling techniques use windowing models and divide the traffic into successive windows to limit the number of packets to be analyzed. There are two main windowing models: fixed and sliding [23]. Using a fixed window, the window boundaries are absolute. The traffic stream is partitioned into non-overlapping windows and the offset between two consecutive windows is equal to the number of packets in the window. Using the sliding window model, the window boundaries are updated over time: when a new packet arrives, it will be added to the window and the oldest one will be removed. In this case, the shift between two consecutive windows is less than the window size, most often equals 1. In this benchmark, two types of estimations will be considered: total traffic statistics and instantaneous traffic statistics estimation. With the total traffic statistics estimation, the overall traffic behavior is predicted. The traffic stream is divided into successive fixed (non-overlapping) windows, then, the sample of the traffic stream is constructed by combining all the sub-samples built over all the fixed windows. The network behavior is analyzed based on the final sample. However, with the instantaneous traffic statistics estimation, the sample is built over the most recent packets of the stream. Thus, the network behavior is analyzed over each sliding window, along the measurement process. In our work, the sampling policies used to estimate the overall traffic statistics are called "non-stream sampling algorithms", while the sampling policies used to estimate instantaneous traffic statistics are called "stream sampling algorithms".
- Sample size: The sample size is most often proportional to the length of the traffic and depends on the sampling ratio. The higher the sampling rate, the higher the accuracy of the sample, nevertheless, this requires more computational resources. Notice that some sampling policies have a fixed and bounded sample size independent of the sampling ratio. In this work, and without loss of generality, all the algorithms will be adapted to give a sample with a ratio-dependent size.

In the following, we discuss in detail each one of these

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3118605, IEEE Access

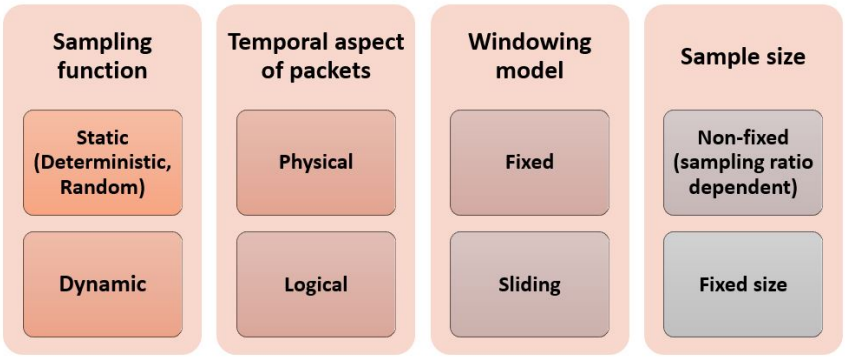


FIGURE 2: Taxonomy of sampling policies.

TABLE 2: Classification of static sampling policies.

Sampling policy	Sampling function		Temporal aspect		Windowing model		Sample size	
	Deterministic	Random	Physical	Logical	Fixed	Sliding	Fixed	Ratio-dependent
SRSFW		✓		✓	✓			✓
SRSSW		✓		✓		✓		✓
DETFW	✓			✓	✓		✓	
DETSW	✓			✓		✓	✓	
Chain-sample		✓		✓		✓		✓
Chain+		✓		✓		✓		✓
Stratified		✓		✓	✓			✓
Systematic	✓			✓	✓		✓	
Reservoir		✓		✓		✓	✓	
Backing		✓		✓		✓		✓
Priority		✓	✓			✓		✓
Random Pairing		✓		✓		✓		✓
WRS-N-P		✓		✓	✓			✓
WRS-N-W		✓		✓	✓			✓
StreamSamp		✓		✓	✓		✓	

sampling policies.

A. SIMPLE RANDOM SAMPLING (SRS) OVER A SLIDING WINDOW (SRSSW)

The SRS algorithm [24] aims to construct a random sample. It samples packets randomly and all packets have the same probability p of being sampled. SRS can be with replacement and without replacement. When applying the SRS with replacement, the sample will contain redundant packets since each packet may be selected at least once. However, with the SRS without replacement, each packet can be sampled only once. In this study, we are concerned with the SRS without replacement.

B. SIMPLE RANDOM SAMPLING (SRS) OVER A FIXED WINDOW (SRSFW)

The SRS without replacement algorithm can be also applied to build a sample over a fixed window. This is done by constructing a sample over each window of the traffic stream and removing the samples constructed on the former windows. To sample k packets from a window of size n, each packet is selected with a probability p equal to the sampling ratio k/n. This step must be repeated until the selection of k distinct packets [20].

C. DETERMINISTIC SAMPLING OVER A FIXED/SLIDING WINDOW

The deterministic algorithm is a non-probabilistic sampling algorithm that constructs a sample without randomness. It consists of constructing a sample of size k by selecting one packet from every x packet of the traffic stream. Assuming that the traffic stream consists of packets with an always-increasing index, to construct a sample of distinct packets among the n most recent packets of the traffic, and given the sampling ratio p, each $1/x$ packet is sampled. The value of x is equal to $100/p$. For instance, if p equals 20%, then, the value of x will be equal to $100/20 = 5$, and thus, every $1/5$ packet will be selected exactly. The selection of one packet from every x packets depends on the packet index. If the packet index equals $\alpha \times n/k$ where α is a positive integer,

the packet will be selected.

D. SYSTEMATIC SAMPLING

Let k be the sample size and n the window size, the systematic sampling algorithm partitions the traffic into x groups, each of size $x = n/k$. Thereafter, it selects a random value $j \in [1, x]$ and adds the packets at the following indexes to the sample: j, $j + x$, $j + 2x$, $j + 3x$, etc. [24].

The deterministic and systematic sampling algorithms have several advantages, the samples are easy to be built and they are faster more than the SRS algorithm. However, the drawback of these algorithms is that the sample lacks randomness. The packets are periodically sampled. If the periodicity of the traffic stream is close to the size of the sample k, the constructed sample will be skewed and not representative of the original traffic.

E. STRATIFIED SAMPLING

The Stratified sampling algorithm [24] divides the traffic into homogeneous subgroups and then, builds, randomly, a sample from each subgroup. Compared to the SRS, the stratified sampling algorithm enhances the sampling accuracy since it ensures a high level of representativity of the whole traffic. Using the stratified sampling algorithm is beneficial in many cases, especially, when it is required to highlight a specific set of packets within the traffic. Stratified sampling can also be applied to ensure the representation of extreme or rare groups of packets in the sample.

F. WEIGHTED RANDOM SAMPLING WITHOUT REPLACEMENT

An effective sample must represent the whole traffic stream. However, this requirement may not be satisfied: some packets may be over-sampled or under-sampled. Consequently, the statistical information inferred from the constructed sample will not be reliable. To deal with the lack of representativeness of some packets in the sample, a correction of the sample is needed. This can be done using the Weighted Random Sampling (WRS) algorithm by sampling each packet with a probability based on the packet's weight [25], [26].

Efraimidis et al. [25], [26] proposed two WRS algorithms. WRS-N-P adds each element e_k to the sample with a probability proportional to the weight w_k of the element, as follows:

$$p_k = \frac{\alpha \times w_k}{\sum_{i=1}^{k} w_i} \tag{1}$$

where α is the sample size.

In turn, WRS-N-W adds each packet e_k to the sample with a probability proportional to the item's weight w_k and relative to the weights of the non sampled items. The sampling probability p_k is calculated as follows:

$$p_k = \frac{w_k}{\sum_{i \in v-S} w_i} \tag{2}$$

where S represents the sample.

G. RESERVOIR SAMPLING

The Reservoir sampling algorithm [27], [28] retains a uniform and random sample of a fixed size of k from the whole stream. At first, the algorithm selects the first k received elements of the stream and adds them to the sample. Subsequently, when a new element arrives, it will be sampled with a probability $p = k/i$, where i is the index of the element in the stream, and an element will be removed from the sample randomly.

H. BACKING SAMPLING

The Backing sampling [29], [30] samples the data as follows: at first, the first k elements of the stream are added to the sample. Thereafter, a random number of elements is skipped and the next element is added to the sample with a probability equal to k/n. Another random number of elements is ignored, and so forth.

I. CHAIN-SAMPLE

The Chain-sample algorithm [31] provides, at any time, a random sample of size k selected from the last elements of the stream. It constructs a sample containing one element selected from the last sliding window of the stream. At first, the algorithm samples one element from the first window with a probability equal to $\frac{min(i,n)}{n}$, where n is the window size, and i is the index of the element in the window. Once selected, a successor's index j is chosen at random for the i^{th} element from the elements with indexes $\in [i+1, i+n]$. When the element with index j arrives at the window, a random successor will be also chosen for it. When the element with index i comes out of the window, it will be removed from the sample and substituted by its successor j. To build a sample containing $k > 1$ elements, all the previous steps must be repeated k times.

J. CHAIN+ SAMPLING

To build a sample containing k elements, the Chain+ sampling algorithm [32] builds one sample of size k instead of constructing and maintaining k independent samples each of size equal to 1. The algorithm samples each element in the first sliding window with a probability equal to $\frac{min(i,n)}{n}$, where n is the window size and i is the index of the element in the window, if and only if it is not present in the sample. This process is repeated until sampling k distinct elements.

K. PRIORITY SAMPLING

Babcock et al. [31] introduced the Priority sampling algorithm that constructs and maintains a random sample over a physical sliding window. To construct a sample containing one element, the priority sampling algorithm assigns a random priority $p \in [0, 1]$ for each element, then selects the element with the highest priority in the sliding window. To construct a sample containing k elements, the process must be repeated k times. In the old version of the priority sampling algorithm, the weights are assigned randomly

without inspecting the arrival time of the element. Also, this algorithm suffers from all the problems of the Chain-sample algorithm. These problems will be presented in Section IV. Therefore, in this work, we implemented the new version of the Priority sampling algorithm proposed by [33]. The modified version of the Priority sampling algorithm alters the traditional algorithm by assigning the weights to the packets according to their arrival time, their contents, and their impact on the sample accuracy.

L. RANDOM PAIRING SAMPLING

The Random Pairing (RP) sampling algorithm [34], [35] constructs and retains a random sample over the most recent sliding window of the stream. To achieve this, three values are calculated on each window: the number of expired elements present in the sample, denoted by c_1, the number of expired elements not present in the sample, denoted by c_2, and the number of all expired elements, denoted by d. When a sampled element expires, it will be deleted from the sample. Each new element is added to the sample based on the value of d. If $d = 0$, sampling the new element will follow the Reservoir sampling algorithm [28]. However, if $d > 0$, the new element will be added to the sample with a probability equal to $\frac{c_1}{c_1+c_2}$.

M. STREAMSAMP

StreamSamp [36] algorithm is a progressive sampling technique based on the Simple Random Sampling algorithm. Once received, stream elements are sampled with a predefined sampling ratio. When the sample size is reached, the sample will be stored with an order equal to 0, and a second sample of the same size will be constructed, and so on. As the number of stream elements increases, the amount of samples of order 0 also increases. When this number exceeds a certain limit, StreamSamp merges the two old samples of order 0 into a single sample by performing a simple random sampling of rate $p = 0.5$. The new sample obtained is of order 1, and so on.

A comparison of the presented algorithms is provided by Table 3.

III. METHODOLOGY

This paper focuses on studying existing static sampling techniques. The efficiency of a sampling policy depends on its capability to balance its precision with the computational resources required. This work investigates the statistical effect of sampling the traffic stream and the execution time needed to sample the traffic. Our methodology consists of using the sampling algorithms presented in in this paper to summarize a real traffic dataset. This allows us to understand the behavior of these algorithms. In order to quantify the distortion introduced by the sampling procedures, we consider comparing different statistical metrics. The overall quantification of statistical changes between sampled and unsampled traffic is defined by the Overall Statistic (OS) calculated as follows [37], [38]:

$$OS = \frac{|(\mu_0 - \mu)|}{\mu_0} + \frac{|(med_0 - med)|}{med_0} + \frac{|(std_0 - std)|}{std_0} \quad (3)$$

where μ_0 is the real average value estimated before sampling the traffic, μ is the estimated average value of a traffic calculated after sampling, med_0 and med are the median values of a traffic parameter being estimated for the before and after sampling respectively, and std_0 and std are the standard deviation values of the traffic estimated before and after sampling respectively.

Our experiment in Section IV can be thus summarized as follows:

- Scenario 1 - Without sampling: In this scenario, the mean, standard deviation, and median of the unsampled dataset are calculated.
- Scenario 2 - With sampling: In this scenario, the mean, standard deviation, median, and OS are calculated based on the sampled data. Different sampling strategies with different parameters will be considered. The computational resources and execution time needed for sampling are also calculated. In the end, a comparative analysis of both scenarios is carried out.

The dataset used in this work is the NSL-KDD [39] which has been developed to enhance the KDD CUP 99 dataset. The main concern with the KDD CUP 99 is the huge number of duplicate records in the training and testing subsets, which leads to inaccurate intrusion detection results [39]. In the NSL-KDD dataset, all the redundant records have been removed. The obtained dataset contains about $150K$ records divided into training and testing subsets. The NSL-KDD dataset consists of 41 attributes and includes 22 attack types.

IV. EXPERIMENTS AND RESULTS

In this section, we investigate the impact of different sampling policies and sampling rates to measure the distortion introduced by the sampling process. We aim at isolating a set of features that are more robust (less distorted) to sampling. The specs of our machine are RAM: 8 GB, System disk: 450 GB, and processor: 2.7 GHz Intel Core $i7$.

A. COMPUTATIONAL RESOURCES

The traffic scenario used to evaluate the sampling policies is the training set of the NSL-KDD dataset which consists of 125.974 records with a size of 18.662 MBytes. The computational resources of packets sampling techniques are analyzed according to the execution time which depicts the time spent in summarizing the traffic. A higher sampling ratio leads to more packets selected. Thus, intuitively, the computational resources of sampling algorithms should be proportional to the sampling rates. The execution times of non-stream and stream policies presented in this paper are evaluated in Figures 3 and 4 respectively, based on the sampling ratio and window size (for stream sampling algorithms).

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3118605, IEEE Access

TABLE 3: Advantages and weaknesses of data streams sampling algorithms.

Sampling Algorithm	Advantages	Weak Points
Simple Random (SRSFW [20] and SRSSW [24])	The sample is accurate, convenient, and representative of the entire stream	Biased sample in case of periodicity in the data stream No skewing ability
Deterministic (DETFW and DETSW) [24], [20]	It provides a sample with an exact size The sample is representative of the entire stream when no periodicity is displayed	Biased sample in case of periodicity in the data stream No skewing ability
Chain-sample [31]	It provides a sample with an exact size	Redundant data in the sample No skewing ability
Chain+ [32]	It provides a representative sample with an exact size without duplication	No skewing ability
Stratified [24]	The use of this algorithm is beneficial when it is desired to high- light a specific subgroup within the data and ensure its presence in the sample This algorithm is also used to represent the smallest, extreme or rare subgroups of the data in the sample	Unbounded sample size No policy to choose the sample size No skewing ability
Systematic [24]	The sample is easy to be built The sampling process is fast and accurate since sampled data are spread over the entire stream	Unbounded sample size Biased sample in case of data stream peri- odicity No skewing ability
Reservoir [27], [28]	This algorithm is simple and suitable for streaming environments it is executed in one pas	Recent elements have less chance of being sampled No skewing ability
Backing [29], [30]	This algorithm is suitable for streaming environments it is executed in one pas	Performs several passes over the data No skewing ability
Priority [31]	It provides a sample with an exact size	No policy for the determination and revi- sion of the weights
Random Pairing [34], [35]	The algorithm builds and maintains a uni- form sample of fixed size	No skewing ability
StreamSamp [36]	The algorithm maintains a sample of a fixed size	No policy for the determination and revi- sion of the weights

Figures 3 and 4 confirm the relation between the sampling ratio and computational resources. Since each sampling policy selects the data samples differently, different computational resources are required by each sampling algorithm, even with the same sampling rate.

Results in Figure 3 show that the deterministic and systematic sampling algorithms have the lowest execution time. Regarding the SRSFW and stratified sampling algorithms, the sample may contain duplicated elements. This redundancy happens when the sampling rate is > 0.1, it arises when an element is sampled many times in the same jumping window. This problem becomes more serious when the values k (sample size) and n (window size) are close to each other. In order to deal with the redundancy problem, and to sample exactly k distinct elements from each window, the sampling process on each window should be repeated until an element is selected which is not present in the sample. This process adds significant overhead in terms of runtime, mainly, when the values of k and n are close to each other.

Figure 5 shows the collision rate of the SRSFW and stratified sampling algorithms and the theoretical probability of collision for sampling k elements from a window of size n. Different sampling rates k/n are used. The window size is fixed to 10 packets. The theoretical probability of collision $P_{collision}$ is computed as follows:

$$P_{collision} = 1 - P_{Selecting\ k\ distinct\ items} = 1 - \frac{n!}{n^k(n-k)!} \quad (4)$$

Figure 5 shows that when the sampling ratio (k/n) is equal to 50%, 30% of the packets in the constructed sample are redundant, which will greatly affect the accuracy of the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3118605, IEEE Access

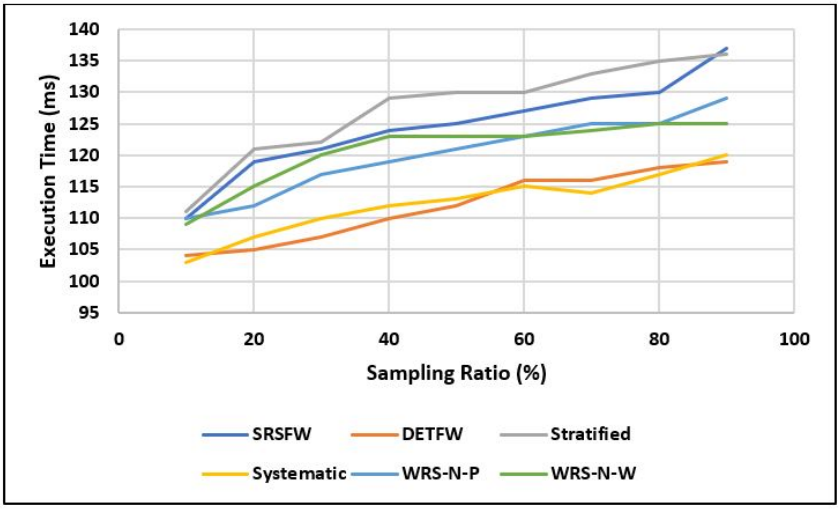


FIGURE 3: Execution time of non-stream sampling algorithms.

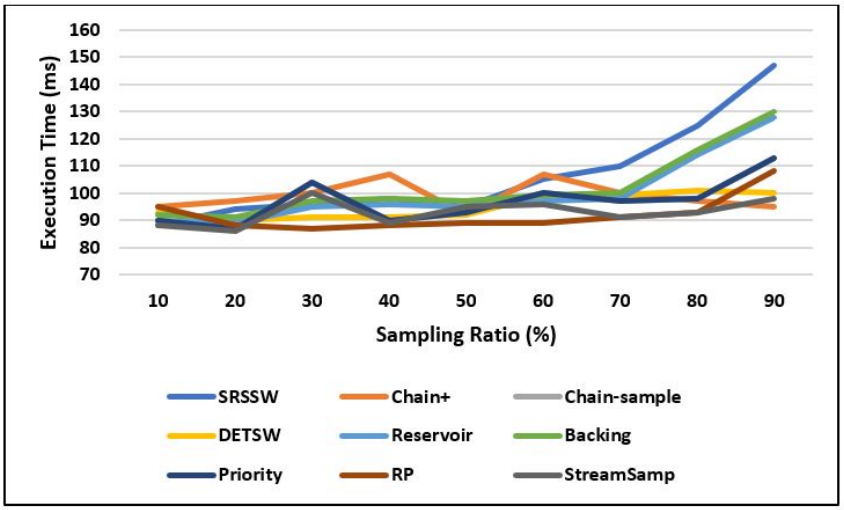


FIGURE 4: Execution time of stream sampling algorithms, using a window size = 10.

sample. Thus, the need to remove the duplicated elements. Figure 5 and Equation 4 show that for a given sampling ratio, when the values of k and n increase, the collision rate also increases. Figure 5 also shows that the collision rate of the SRSSW algorithm is a little bit higher than that of the stratified sampling algorithm. In fact, with the stratified sampling algorithm, each packet in the traffic stream must be added to a subgroup before sampling. After the subgroups are formed, some elements of each subgroup are selected. Thus, the collision rate will be lower. Due to this process, building the sample using the stratified sampling algorithm is more expensive than with the SRSWF algorithm.

Figure 4 shows the execution time of the stream sampling algorithms described in this paper. The window size is fixed to 10 packets. Results show that, like the SRSFW, the execution time of the SRSSW increases when the sample size k and the sliding window size n are close to each other, mainly, when the value of k/n is close to 1. This problem also occurs because of the redundancy issue that arises when the sampling rate is > 0.1. Like the SRSFW algorithm, and in order to avoid duplication in the sample, the SRSSW selection process is repeated until an element is selected which is not present in the current sample. Therefore, a considerable additional cost in terms of execution time is added, mainly, when the value of k/n is high. Even if the SRSSW and Chain+ sampling algorithms use very similar sampling procedures, results in Figure 4 show that the difference in execution time for these two algorithms increases as k/n increases, and becomes clearer when k/n is > 0.5. In fact, the Chain+ sampling algorithm minimizes the collision rate to be equal to that of $k/n = k/n - 0.5$ when k/n is > 0.5, as proven in [32], which decreases the execution time of this algorithm. One can also notice that the priority and chain-sample algorithms have almost the same execution time.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3118605, IEEE Access

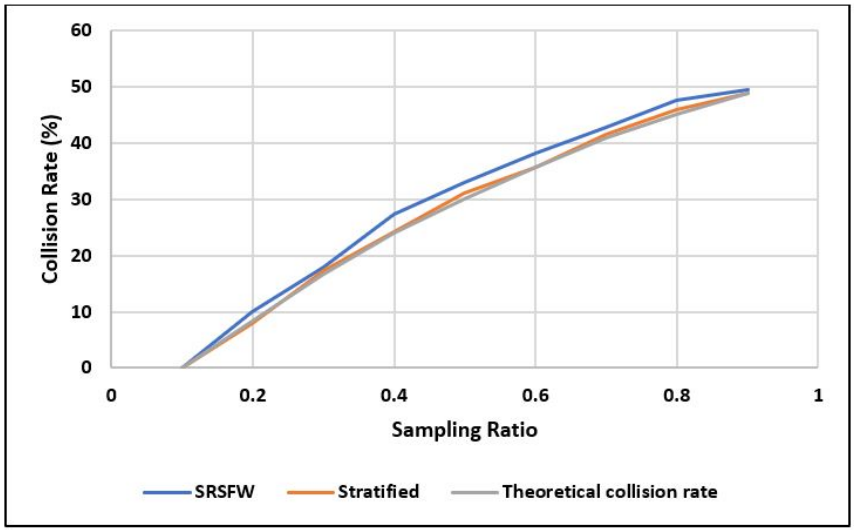


FIGURE 5: Collision rate of the SRSFW and stratified sampling algorithms.

Figure 6 presents the execution time of the stream sampling algorithms for various sampling rates and sliding window sizes. Three window sizes are considered: 10, 100, and 1000. The results show a clear trade-off between the execution time of all these algorithms and the window size: the execution time becomes longer for a larger window. In fact, varying the window size leads to different sample sizes and, therefore, to different computational requirements, even with the same sampling rate. For instance, to sample 20% of the most recent traffic leads to a data volume equal to $2/10$, $20/100$, and $200/1000$ for a window size equal to 10, 100, and 1000 respectively, the required execution time using the SRSSW is 88, 95, and 111 milliseconds respectively. This behavior is observed for all stream sampling policies. It is also observed that there is a stabilization of the execution time for the DETSW algorithm, as shown in Figure 7. This algorithm has the less variation in the execution time whatever is the sampling ratio, while the SRSSW exhibits the highest variation.

B. ESTIMATING TOTAL TRAFFIC STATISTICS

Several monitoring and management activities are carried out using the measurement of the network. Therefore, and despite the importance of reducing the execution time of the sampling process, the sampling policy should describe the behavior of the network accurately. In this section, we analyze and compare the ability of each sampling policy in providing accurate estimations regarding the traffic characteristics.

Our work in this section consists of reducing the size of the traffic using all the sampling policies described previously and then, evaluating the statistical measures of the traffic. Our benchmarking study considers the accuracy of the sampling process regarding the unsampled (original) traffic stream, while also comparing each sampling policy with the others. All sampling policies are evaluated on their ability to provide samples that accurately represent traffic behavior.

The methodology we adopt in this section is to sample the NSL-KDD traffic using all the sampling techniques described previously and then, calculate several statistical measures to assess the accuracy of the sampling estimates for the unsampled traffic while comparing each sampling policy with the others. To achieve this goal, the mean, standard deviation, and median of traffic stream are estimated before and after sampling, as shown in Figures 8 and 9. The sampling accuracy is also evaluated through the Overall Statistic (OS), which represents the relative error of the estimated mean, median, and standard deviation regarding those of the original traffic, as detailed Section III.

Table 4 presents the most important numeric features of the NSL-KDD dataset that can be used to detect the DoS, Probe, R2L, and U2R attacks, according to Ao et al. [40].

1) DoS attack

Figures 10.c, 11.c, and 12.c show the variation of the OS metric based on the estimated mean, standard deviation, and median values of features 5, 7, and 8 using sampled data. Different non-stream sampling policies and various sampling rates $\in [10, 90]$ are considered. Results in these figures show that, whatever is the non-stream sampling policy used, the level of distortion introduced by the sampling process decreases significantly when the sampling ratio increases. In fact, when the interval in which packets are sampled from the network increases, significant periods of network activity will be considered by the sampling process. Results also show that, for a given sampling strategy, the estimated OS metric for features 7 and 8 does not vary significantly when the sampling ratio is $\in [70, 90]$. For feature 5, the variation of the OS metric is small when the sampling ratio is $\in [80, 90]$. All these results demonstrate that sampling fewer packets (less than 90%) does not lead to less accurate estimation.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3118605, IEEE Access

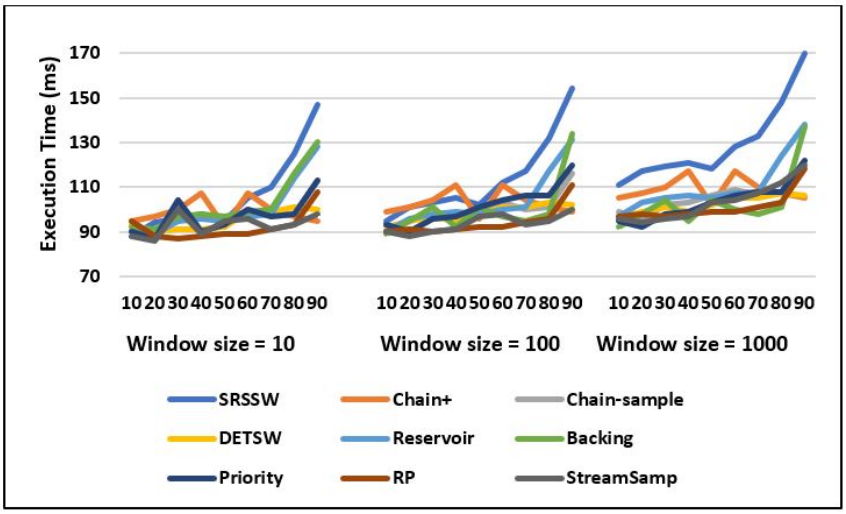


FIGURE 6: Execution time of stream sampling algorithms for different window sizes.

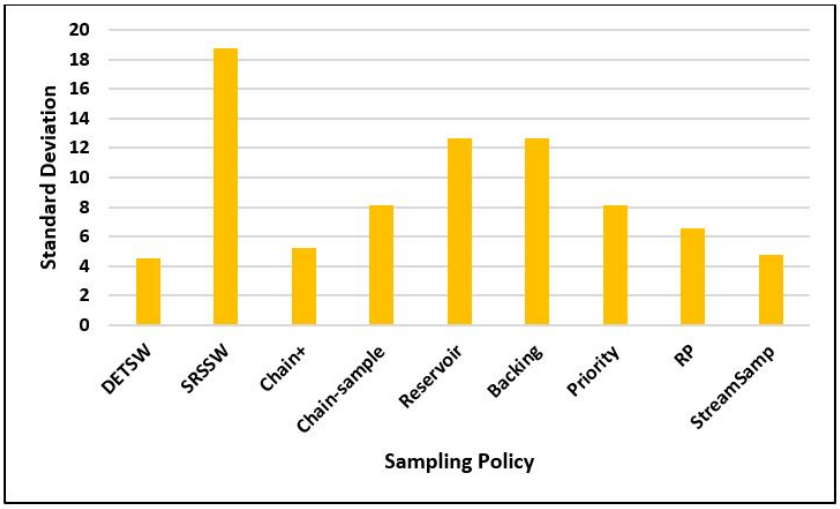


FIGURE 7: Variation of the execution time of stream sampling algorithms, using a window size equal 10.

TABLE 4: Relevant features for each attack type in the NSl-KDD dataset.

Attack	Features names	Features numbers
DoS	source bytes, land, wrong fragment	5, 7, 8
Probe	source bytes, srv error rate, diff srv rate, src port rate	5, 28, 30, 36
R2L	destination bytes, failed logins, count, dst host error rate	6, 11, 23, 39
U2R	root shell, srv count, src port rate	14, 24, 36

FIGURE 8: Traffic statistics estimation for unsampled data.

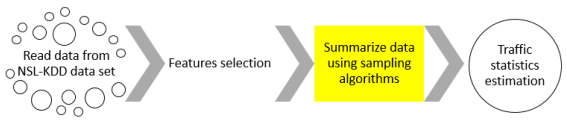


FIGURE 9: Traffic statistics estimation for sampled data.

Thus, there is a possibility to save computational resources by collecting fewer packets and achieving a high sampling

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3118605, IEEE Access

accuracy.

Figures 10, 11, and 12 show in detail the accuracy of the estimated mean, standard deviation, and median of the features 5, 7, and 8, according to the sampling strategy and sampling ratio.

Results in Figure 10 show that when the sampling ratio is $\in [80, 90]$, the accuracy of the mean and standard deviation metrics is approximately the same and very close to the real mean and standard deviation values of the unsampled traffic.

Figure 11 shows that for a sampling ratio $\in [60, 90]$, the level of distortion introduced by the sampling process using the deterministic and systematic sampling is null. Also, for a given sampling ratio less than 50%, the estimation accuracy of all the algorithms is highly variable. Results also show that the WRS-N-W is the worst sampling strategy since it gives the highest OS value whatever is the sampling ratio.

According to Figures 11.c, one can notice that like feature 5, and for a sampling ratio $\in [60, 90]$, the level of distortion introduced by the sampling process using the deterministic and systematic sampling is null. For a given sampling ratio less than 50%, the estimation accuracy of all the algorithms is also highly variable.

According to Figure 12.c, one can notice that like features 5 and 7, and for a sampling ratio $\in [60, 90]$, the level of distortion introduced by the sampling process using the deterministic and systematic sampling is null. For a given sampling ratio less than 50%, the estimation accuracy of all the algorithms is highly variable. Results also show that the SRSFW and WRS-N-W are the worst sampling strategies since they give the highest OS value whatever is the sampling ratio.

Based on the above discussion, one can conclude that to estimate the values of features 5, 7, and 8 needed to detect the DoS attack, the best sampling strategy, and sampling ratio to apply in order to achieve the lowest distortion level is the deterministic/systematic sampling with a sampling ratio equal to 60%. Since the deterministic and systematic strategies require the lowest execution time and computational resources also, there is a possibility to save computational resources by collecting fewer packets and achieving a very high sampling accuracy when using these strategies.

2) Probe attack

Figures 10.c, 13.c, 14.c, and 15.c show the variation of the OS metric based on the estimated mean, standard deviation, and median values of features 5, 28, 30, and 36 using sampled data while considering different non-stream sampling policies and various sampling rates $\in [10, 90]$. Results show that whatever is the non-stream sampling policy used, the level of distortion introduced by the sampling process does not vary significantly when the sampling ratio is $\in [70, 90]$. Results also show that for a given sampling strategy the estimated OS metric for feature 28 does not vary significantly when the sampling ratio is $\in [60, 90]$. For feature 5, the variation of the OS metric is small when the sampling ratio is $\in [80, 90]$. All these results show that sampling fewer packets will not lead to a less precise estimate. Thus, there is a possibility to save computational resources by collecting fewer packets and achieving a high sampling accuracy.

3) R2L attack

Figures 16.c, 17.c, 18.c, and 19.c show the variation of the OS metric based on the estimated mean, standard deviation, and median values of features 6, 11, 23, and 39 using sampled data while considering different non-stream sampling policies and various sampling rates $\in [10, 90]$. Results also show that whatever is the non-stream sampling policy used, the level of distortion introduced by the sampling process does not vary significantly when the sampling ratio is $\in [70, 90]$.

4) U2R attack

Figures 20.c, 21.c, and 15.c show the variation of the OS metric based on the estimated mean, standard deviation, and median values of features 14, 24, and 36 using sampled data while considering different non-stream sampling policies and various sampling rates $\in [10, 90]$. Results also show that whatever is the non-stream sampling policy used, the level of distortion introduced by the sampling process does not vary significantly when the sampling ratio is $\in [70, 90]$. The comparison of the different sampling algorithms shows that in general, the sampling precision is high and less variable when the sampling ratio is $\in [50, 90]$.

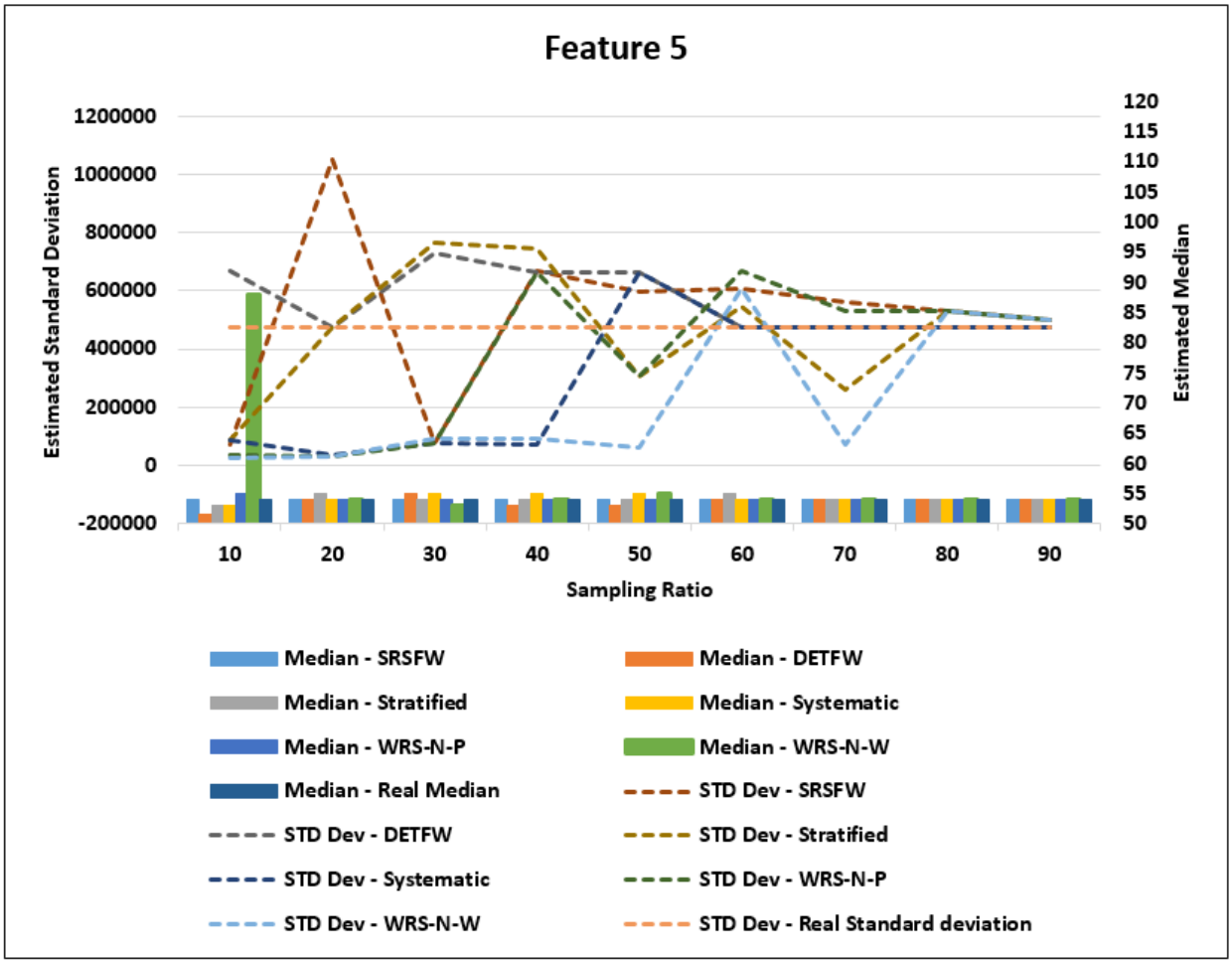
C. ESTIMATING INSTANTANEOUS TRAFFIC STATISTICS

A common way to understand the traffic behavior is to estimate the traffic statistics instantaneously, in a time interval. Thereby, the accuracy in estimating the traffic behavior over time is analyzed through instantaneous mean, median, standard deviation, and OS metric constantly calculated along the measurement process.

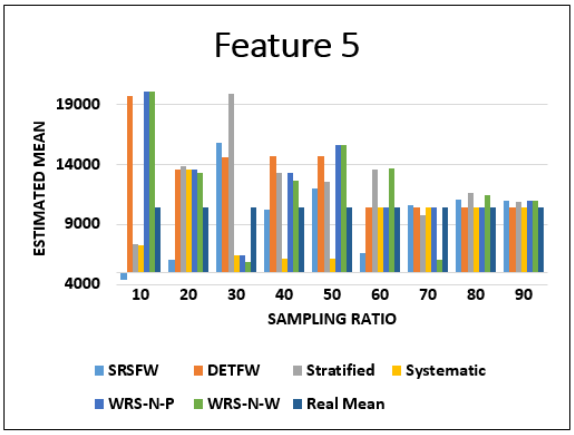
1) DoS attack

Figures 22.c, 23.c, and 24.c show the variation of the OS metric based on the estimated mean, standard deviation, and median values of features 5, 7, and 8 using sampled data. Different stream sampling policies, various sampling rates $\in [10, 90]$, and various window sizes are considered. Results in these figures show that the Chain-sample algorithm has the highest OS value, whatever is the sampling rate ($\in [20\%, 90\%]$) and window size, and the performance of this algorithm in terms of the OS value decreases when the sampling rate increases or/and when the window is large. It is because of the bad quality of the constructed sample since the number of collisions will increase when the value of k/n increases. One can also notice that for a sampling rate equal to 10% and window size equal to 10, the OS value of the Chain-sample algorithm is very close to that of the SRSSW algorithm since the collision rate, in this case, is equal to 0%. When the sampling rate increases, or when the window size increases, the collision rate will be higher, thus, leading to a higher OS metric.

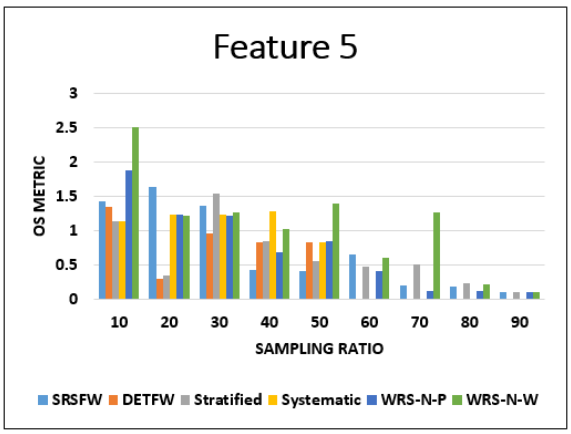
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3118605, IEEE Access



(a) Standard deviation and median estimation



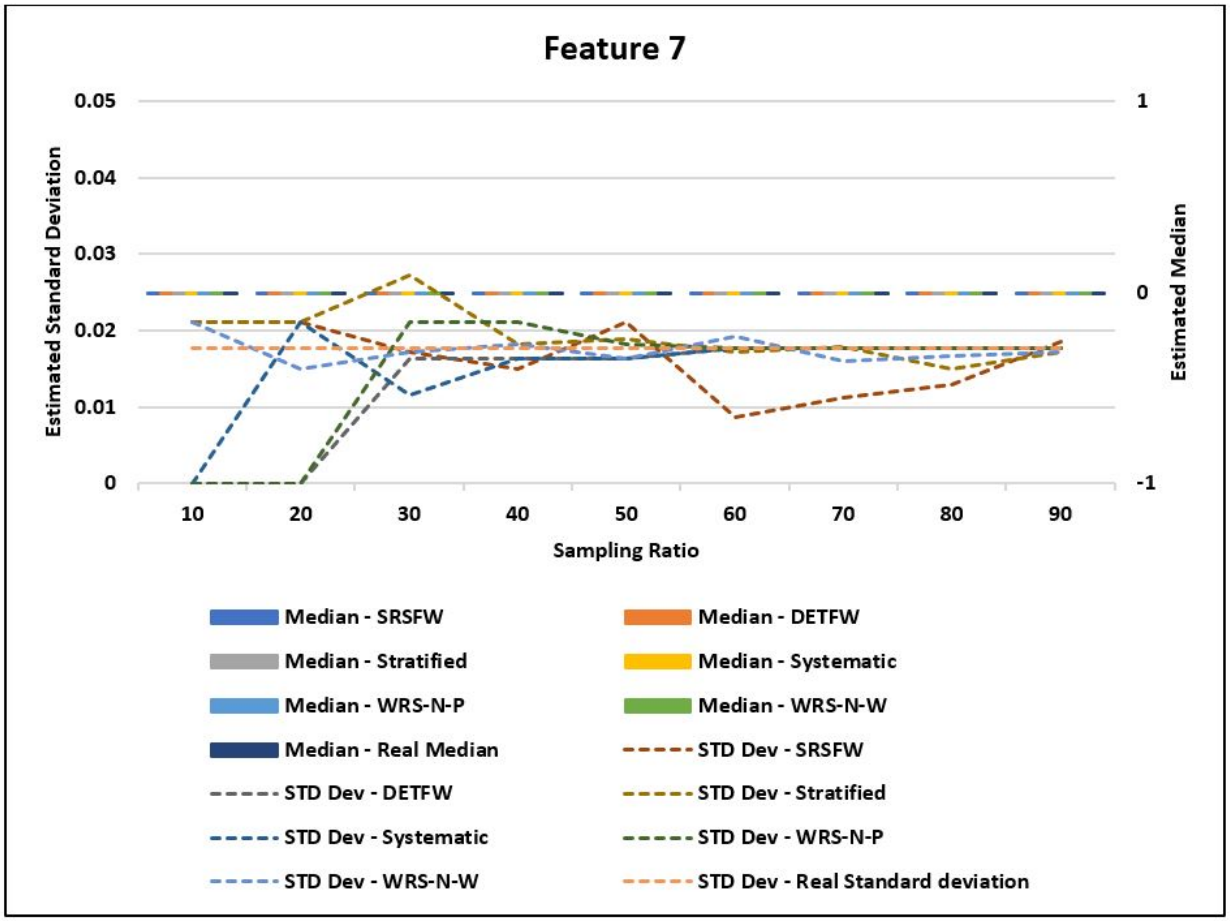
(b) Mean estimation



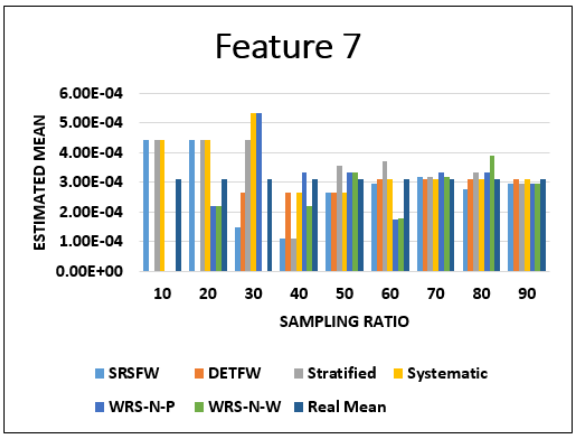
(c) OS metric.

FIGURE 10: Statistical metrics estimation of feature 5 using non-stream sampling policies.

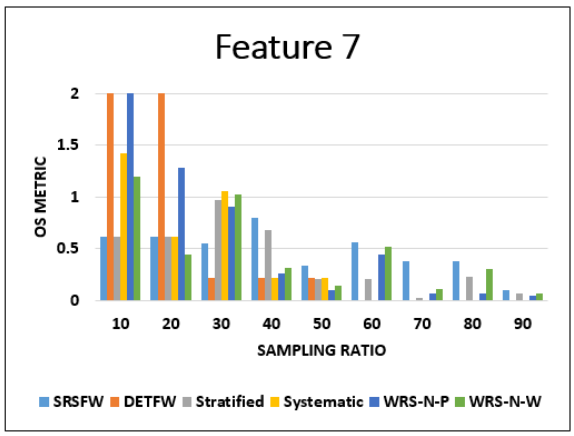
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3118605, IEEE Access



(a) Standard deviation and median estimation



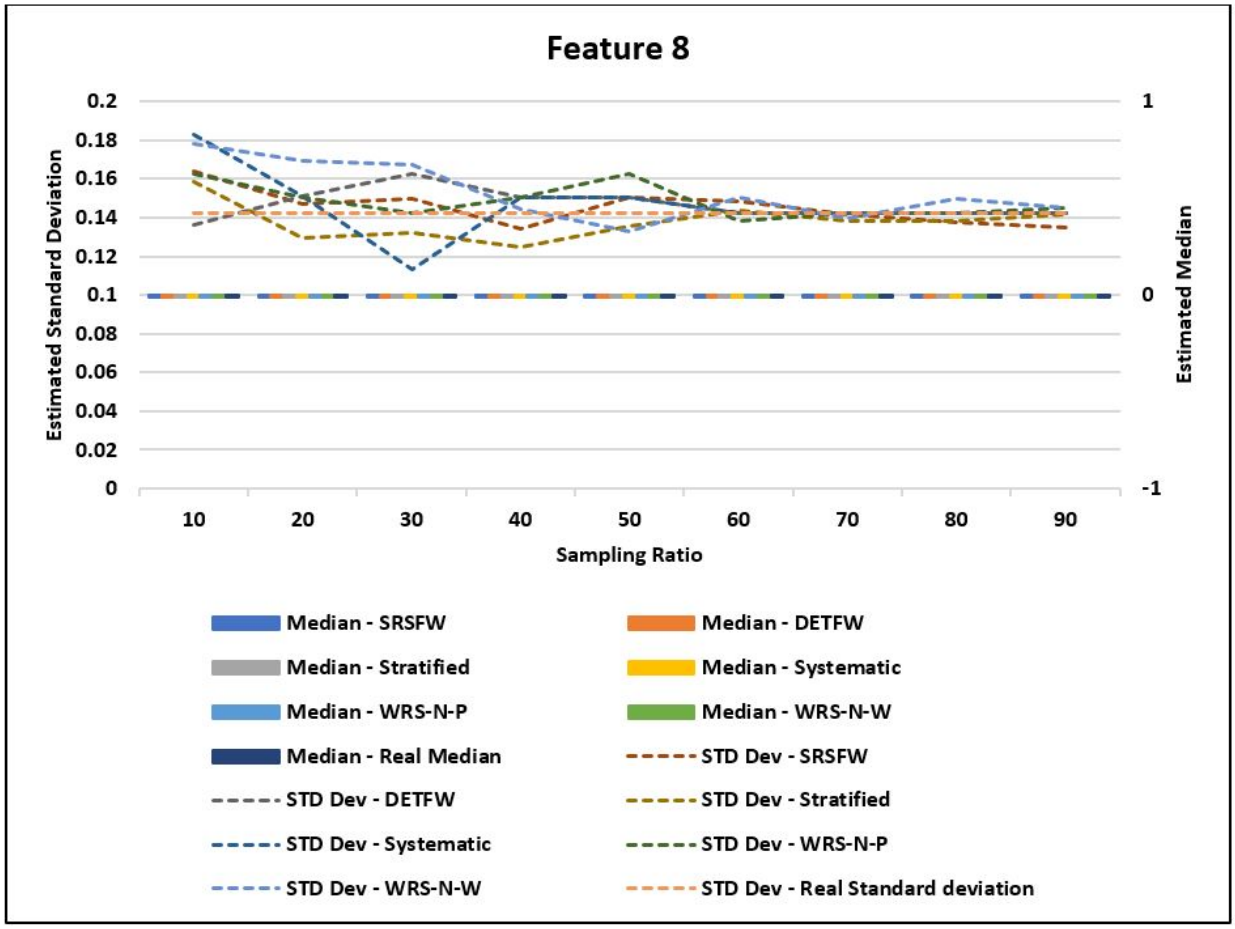
(b) Mean estimation



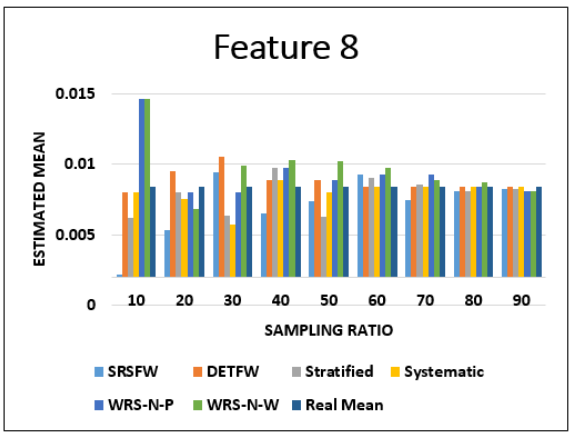
(c) OS metric.

FIGURE 11: Statistical metrics estimation of feature 7 using non-stream sampling policies.

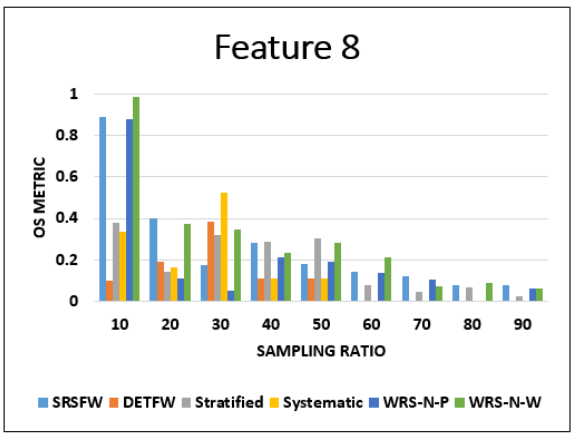
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3118605, IEEE Access



(a) Standard deviation and median estimation



(b) Mean estimation

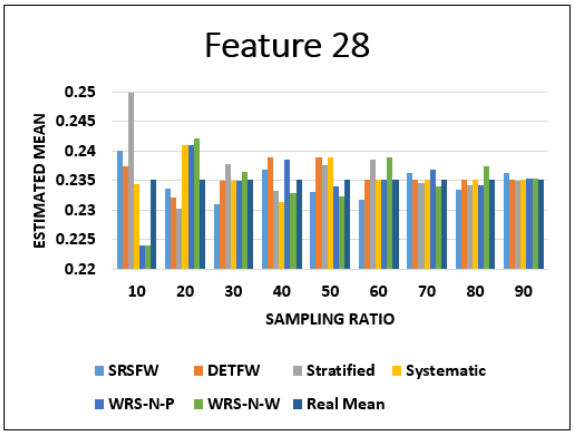


(c) OS metric.

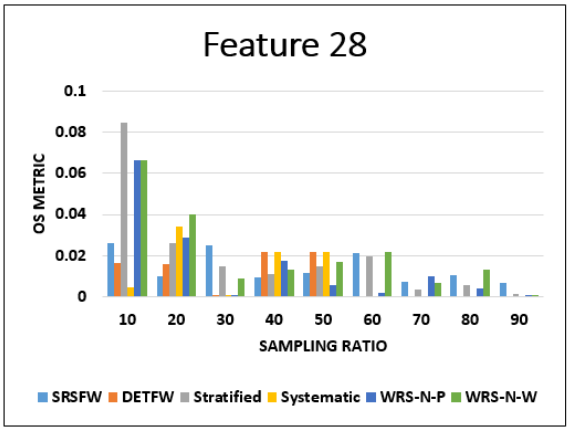
FIGURE 12: Statistical metrics estimation of feature 8 using non-stream sampling policies.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3118605, IEEE Access

(a) Standard deviation and median estimation



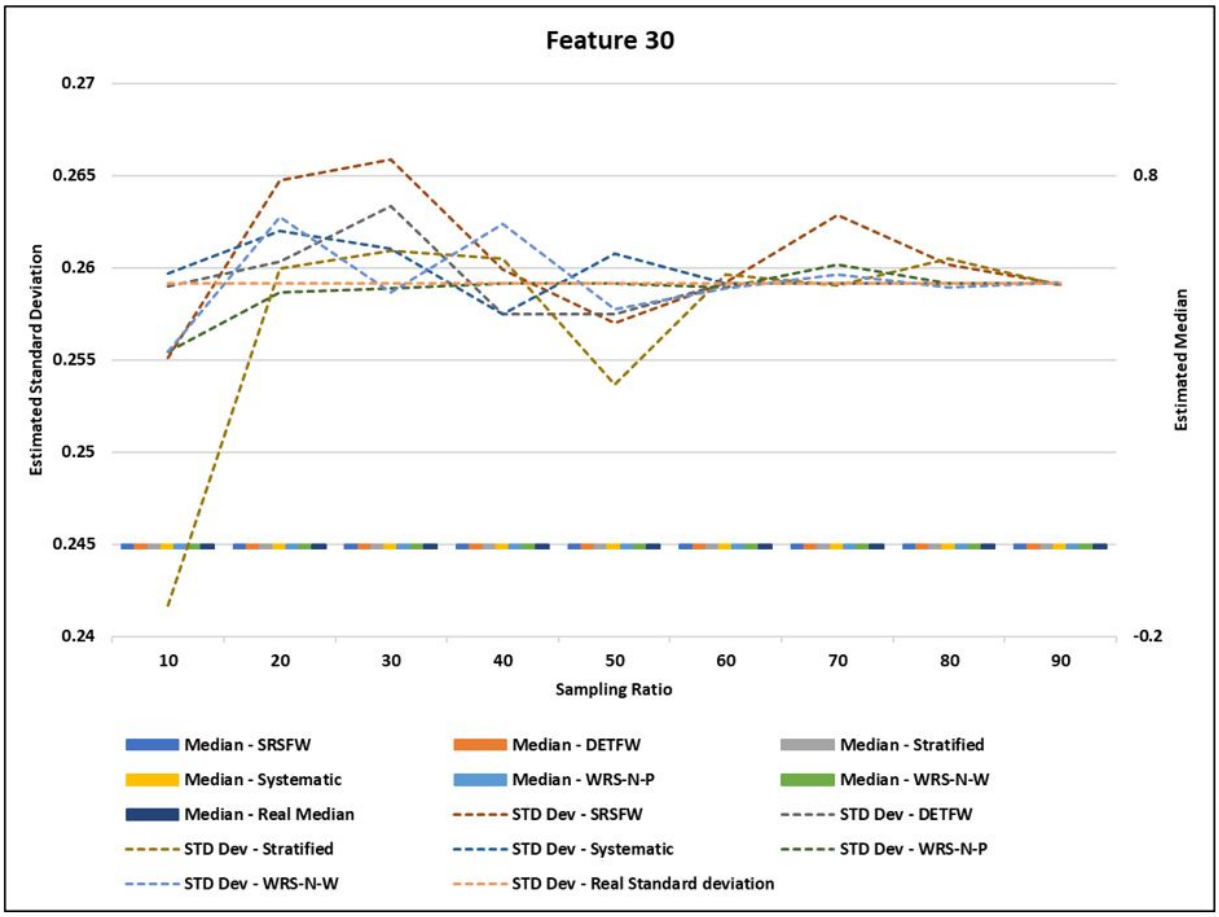
(b) Mean estimation



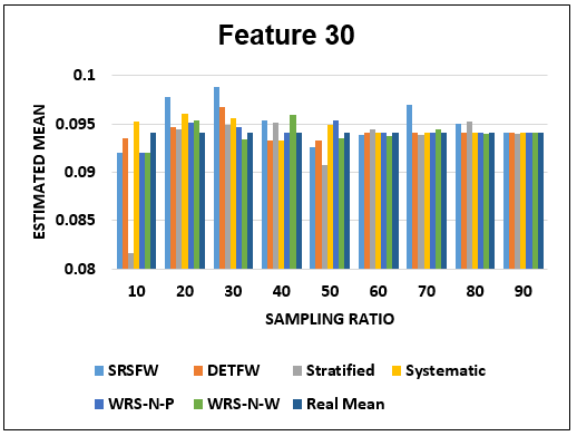
(c) OS metric.

FIGURE 13: Statistical metrics estimation of feature 28 using non-stream sampling policies.

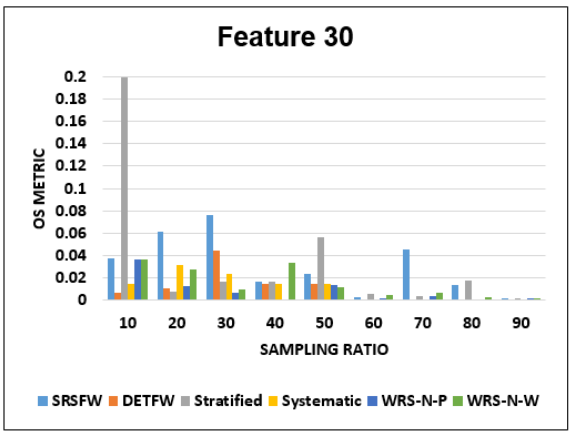
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3118605, IEEE Access



(a) Standard deviation and median estimation



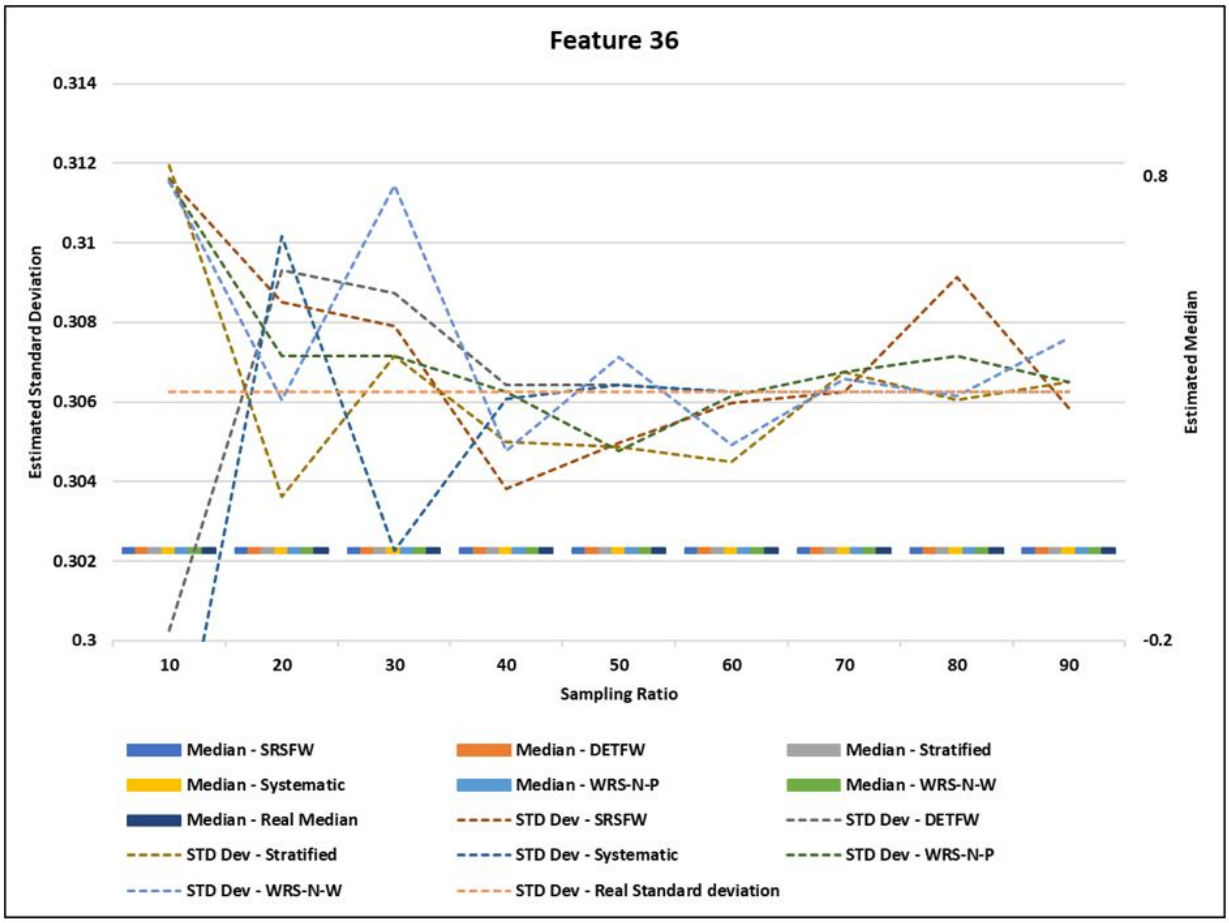
(b) Mean estimation



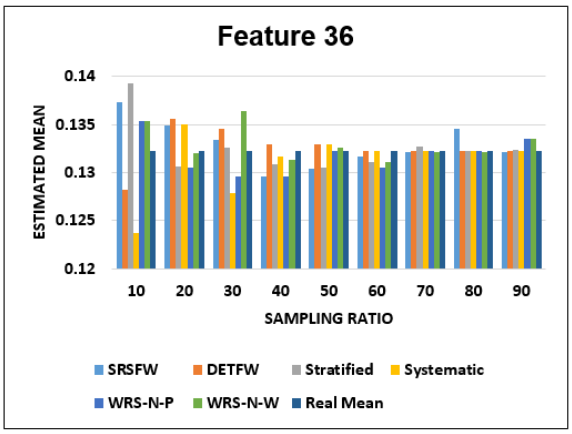
(c) OS metric.

FIGURE 14: Statistical metrics estimation of feature 30 using non-stream sampling policies.

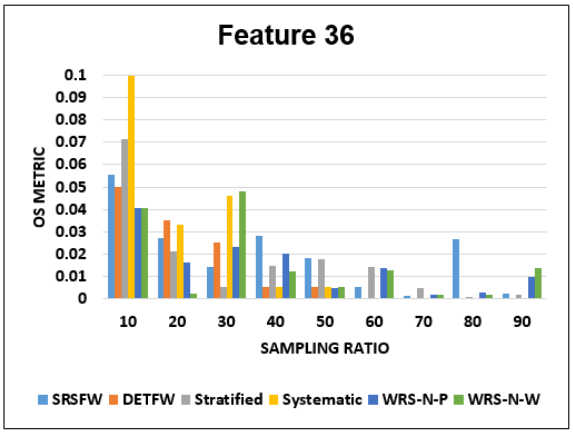
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3118605, IEEE Access



(a) Standard deviation and median estimation



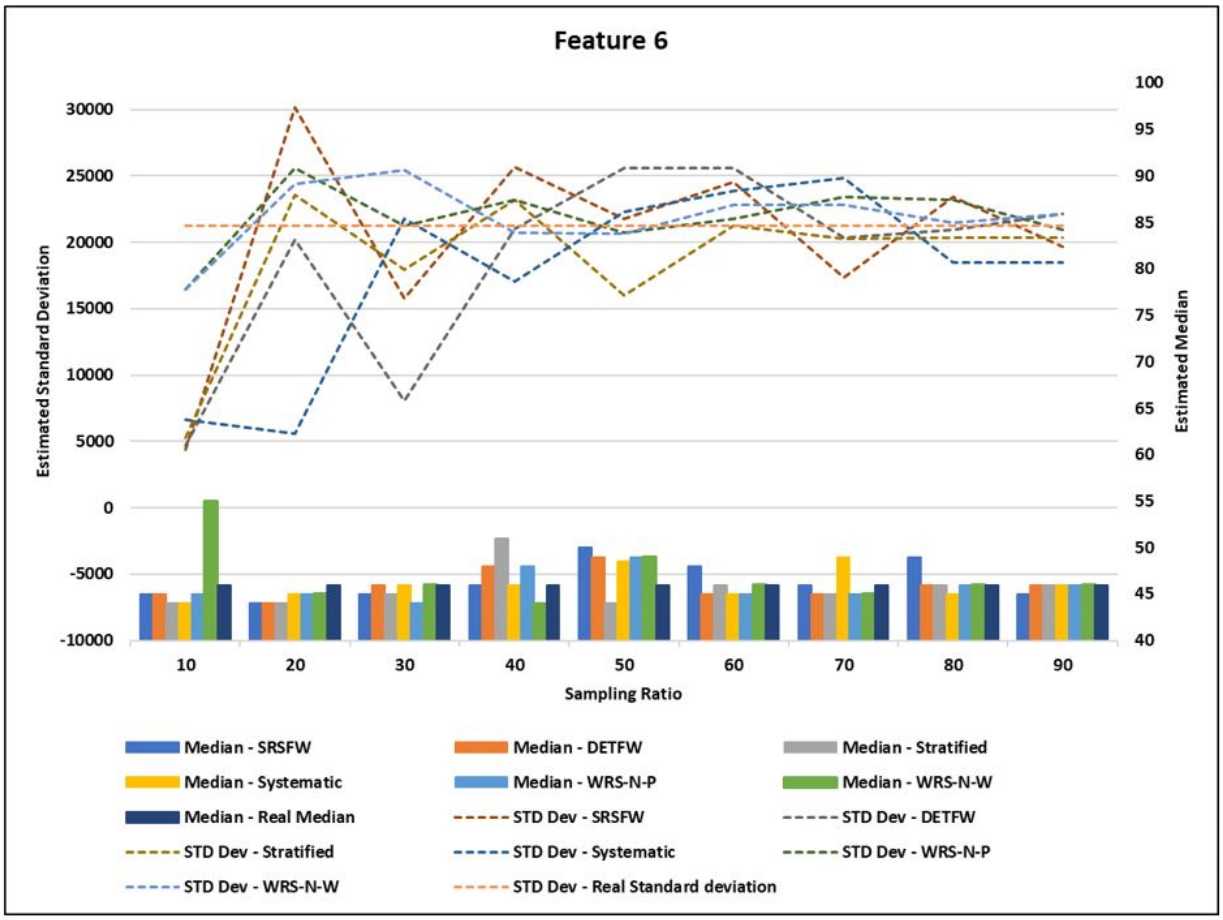
(b) Mean estimation



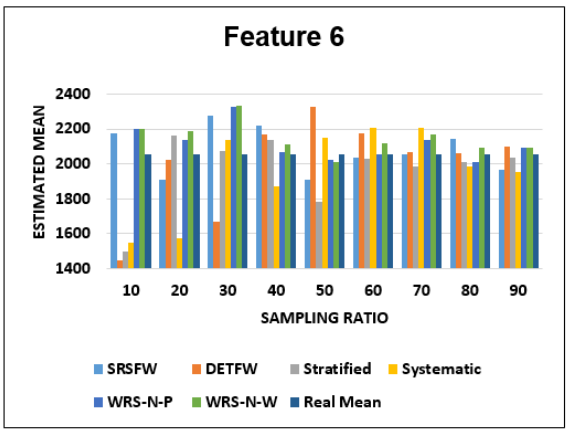
(c) OS metric.

FIGURE 15: Statistical metrics estimation of feature 36 using non-stream sampling policies.

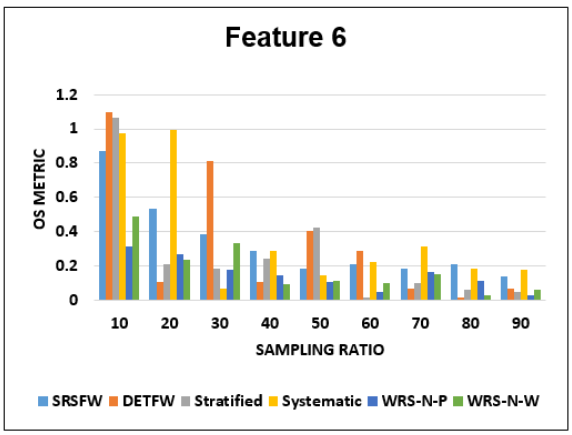
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3118605, IEEE Access



(a) Standard deviation and median estimation



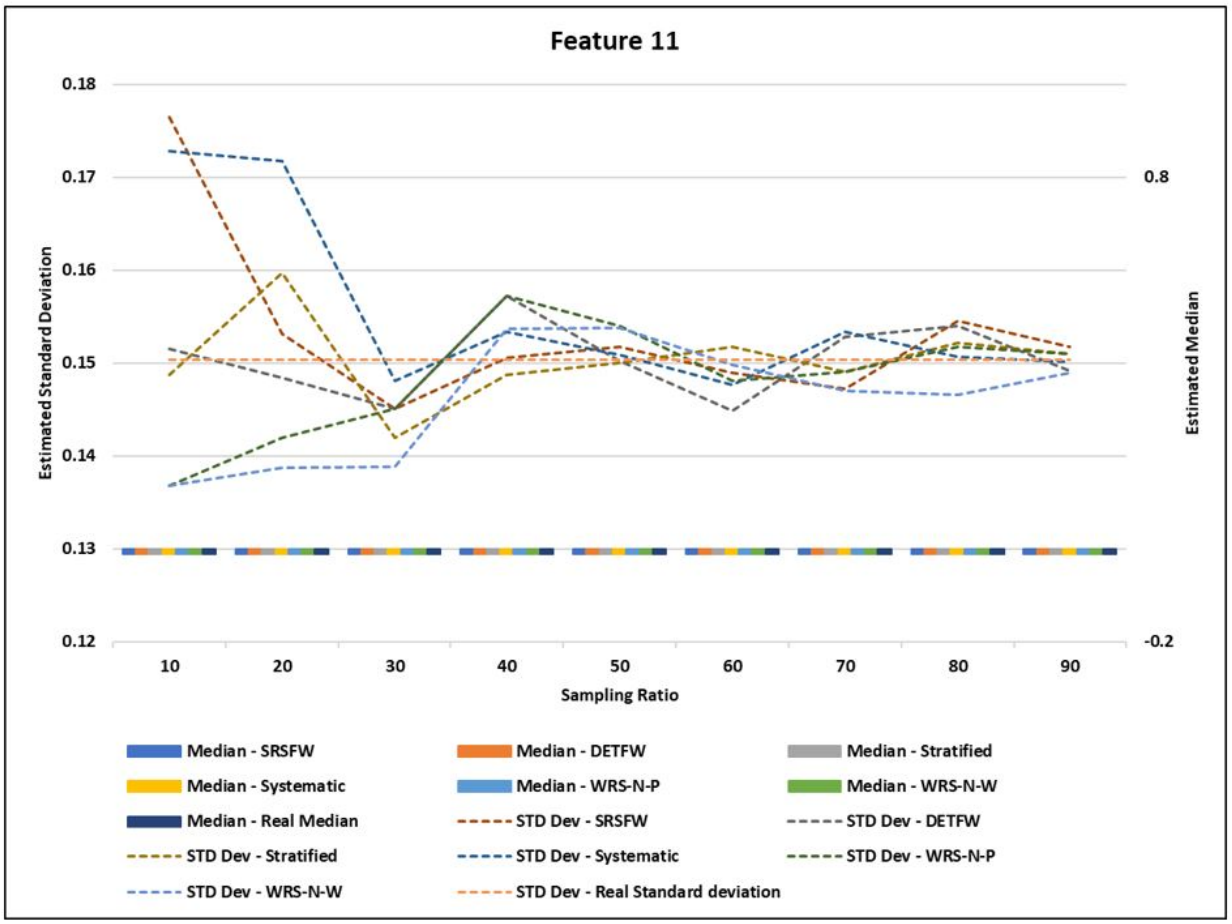
(b) Mean estimation



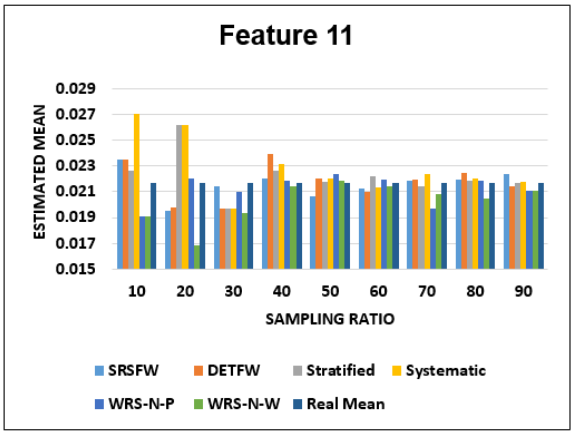
(c) OS metric.

FIGURE 16: Statistical metrics estimation of feature 6 using non-stream sampling policies.

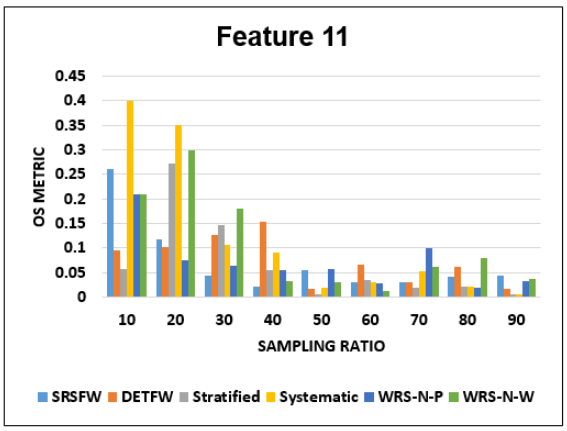
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3118605, IEEE Access



(a) Standard deviation and median estimation



(b) Mean estimation

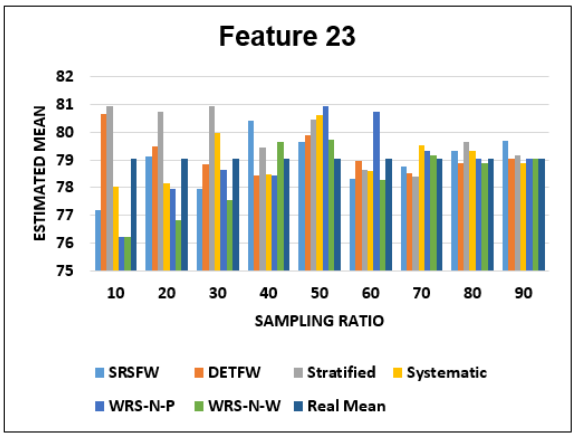


(c) OS metric.

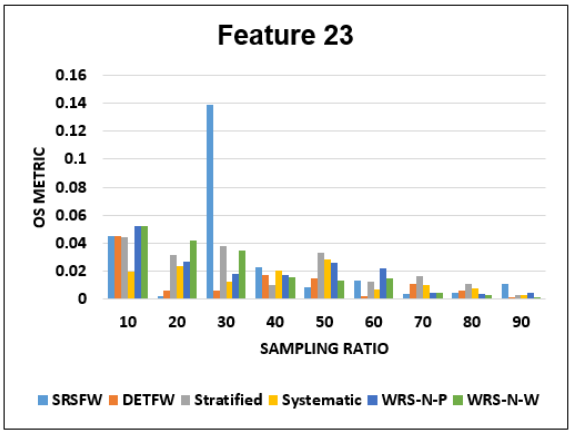
FIGURE 17: Statistical metrics estimation of feature 11 using non-stream sampling policies.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3118605, IEEE Access

(a) Standard deviation and median estimation



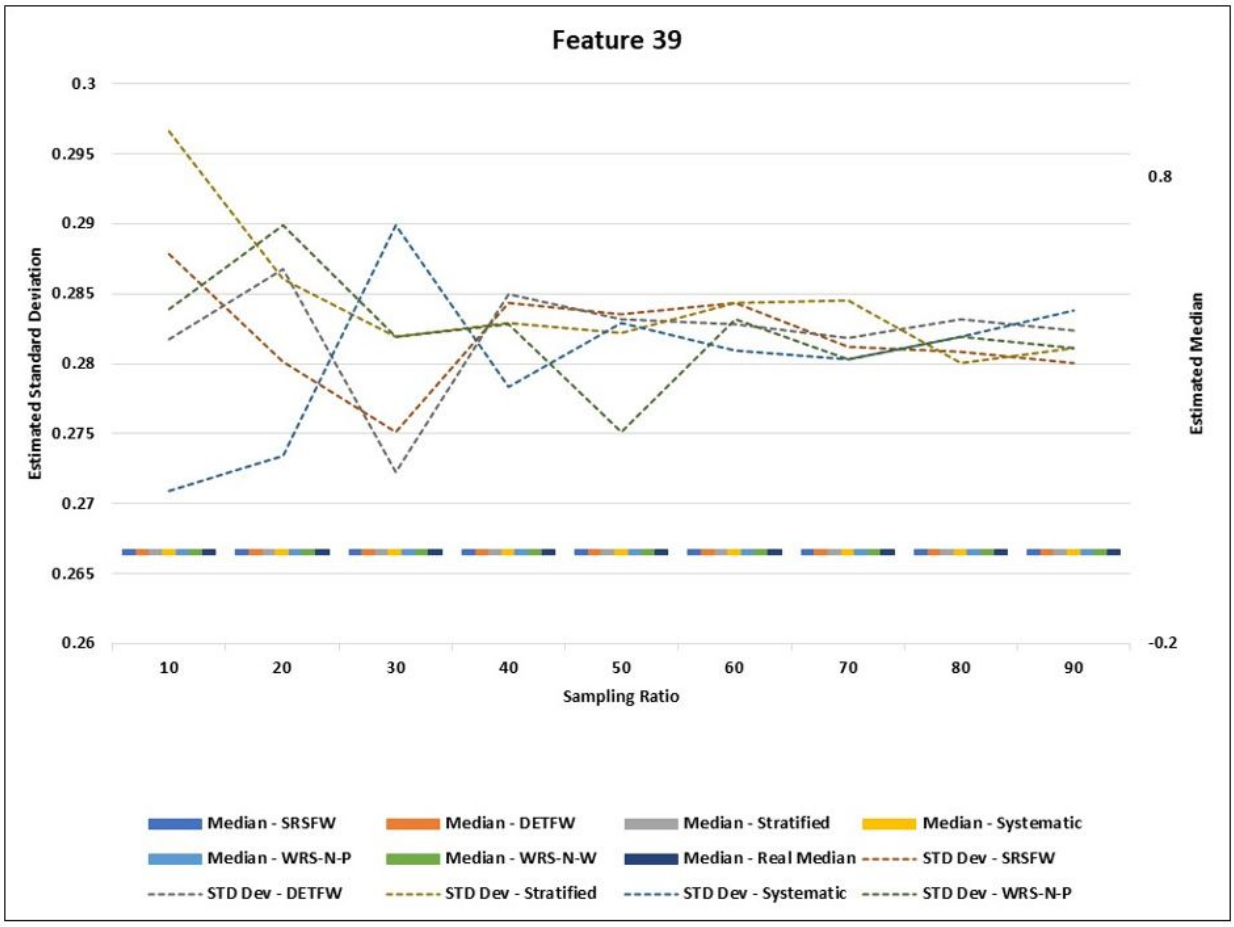
(b) Mean estimation



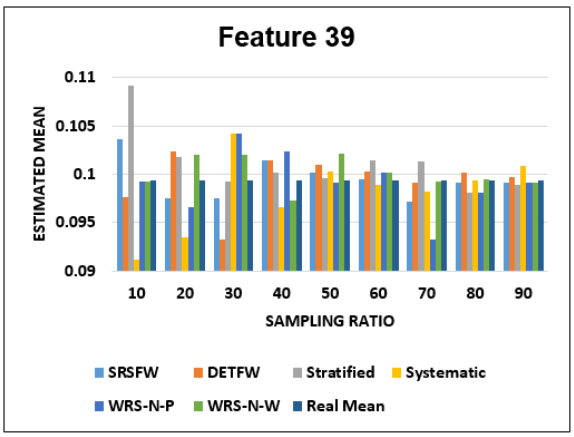
(c) OS metric.

FIGURE 18: Statistical metrics estimation of feature 23 using non-stream sampling policies.

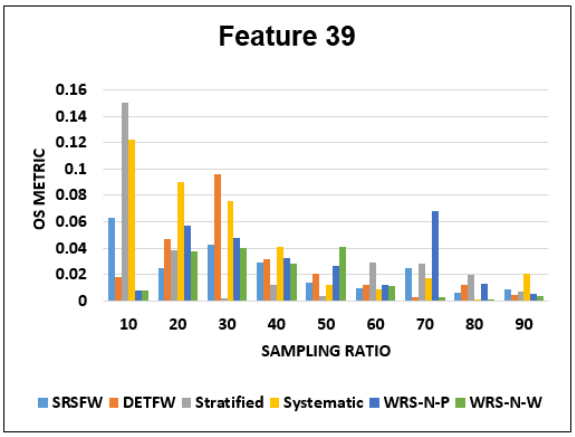
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3118605, IEEE Access



(a) Standard deviation and median estimation



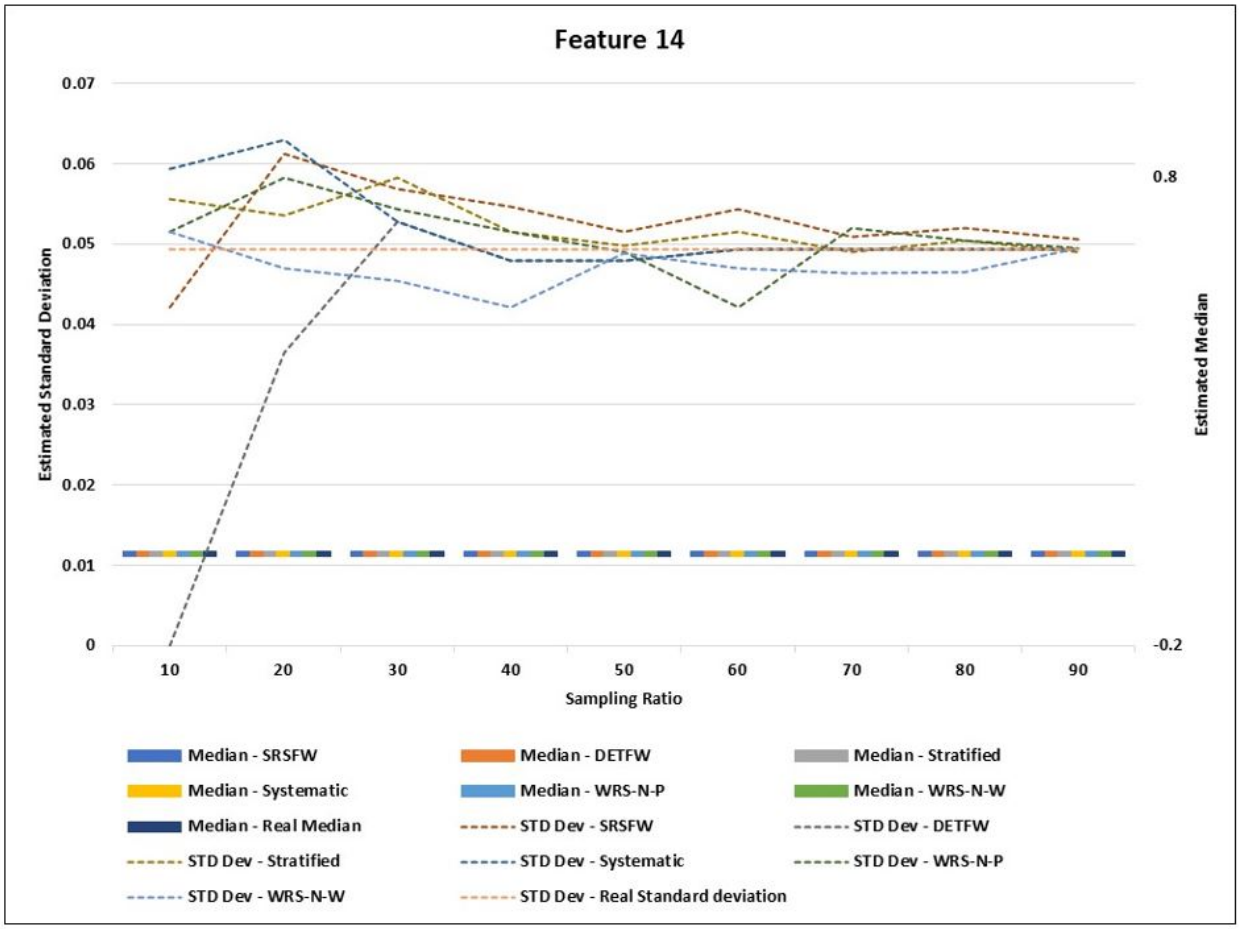
(b) Mean estimation



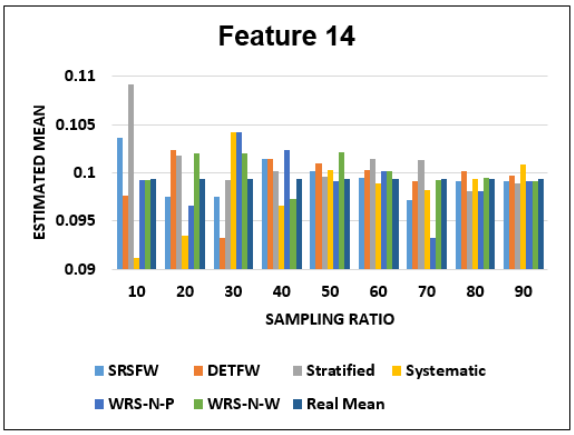
(c) OS metric.

FIGURE 19: Statistical metrics estimation of feature 39 using non-stream sampling policies.

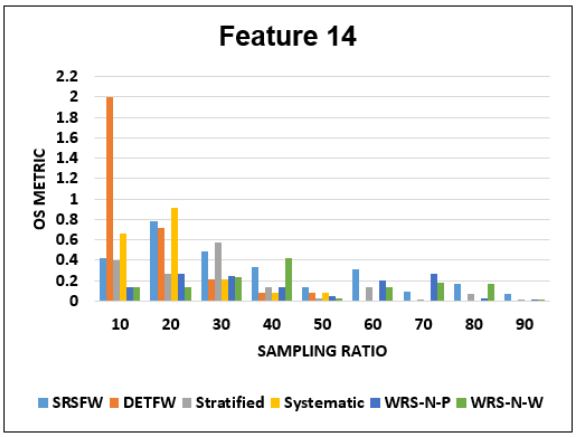
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3118605, IEEE Access



(a) Standard deviation and median estimation



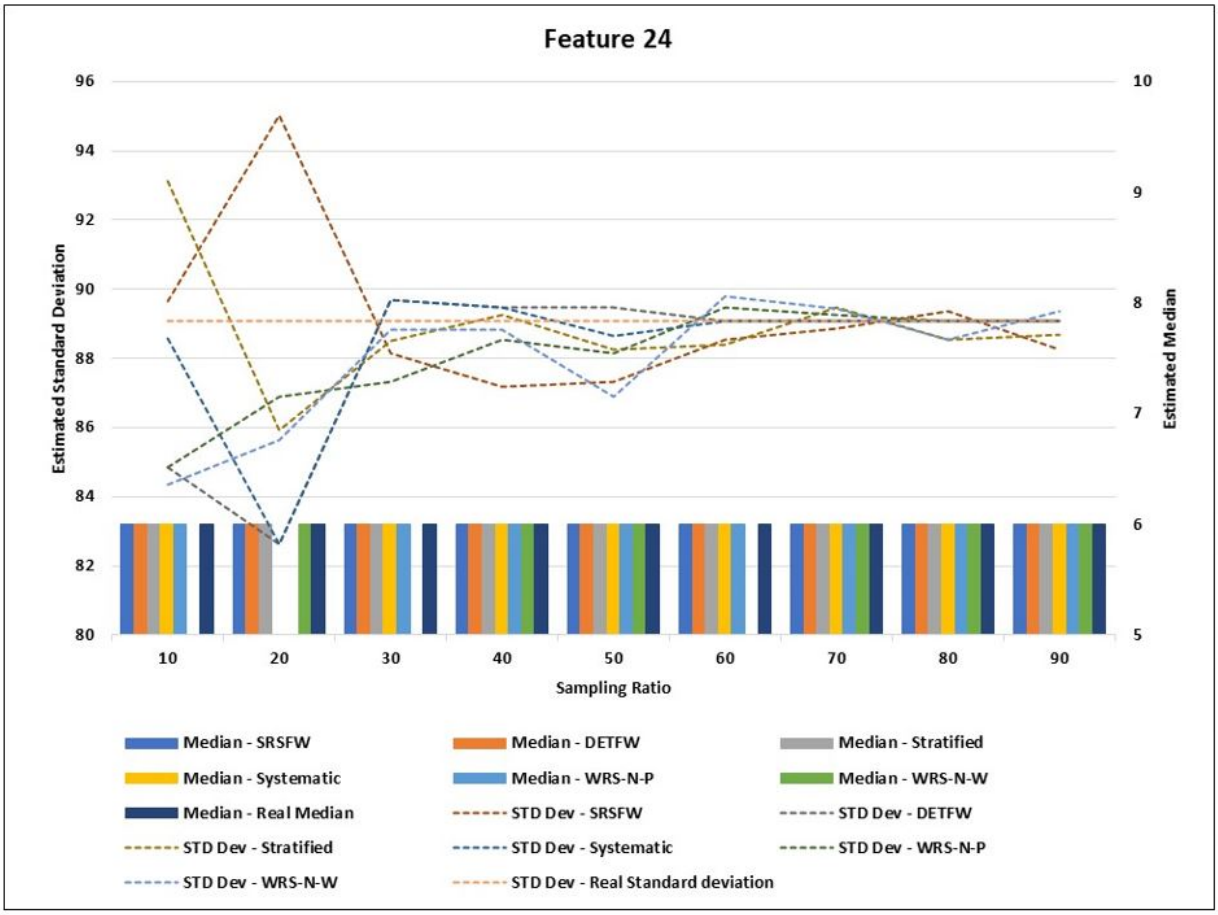
(b) Mean estimation



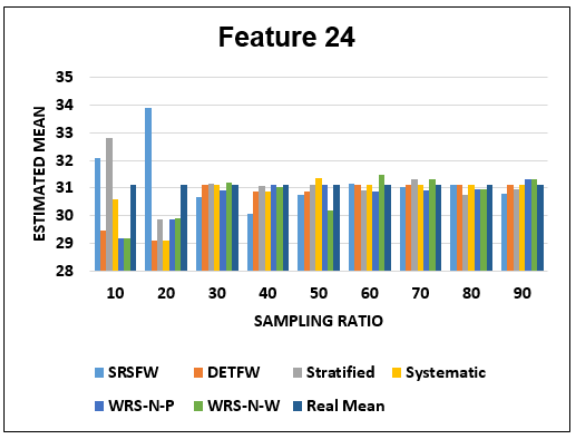
(c) OS metric.

FIGURE 20: Statistical metrics estimation of feature 14 using non-stream sampling policies.

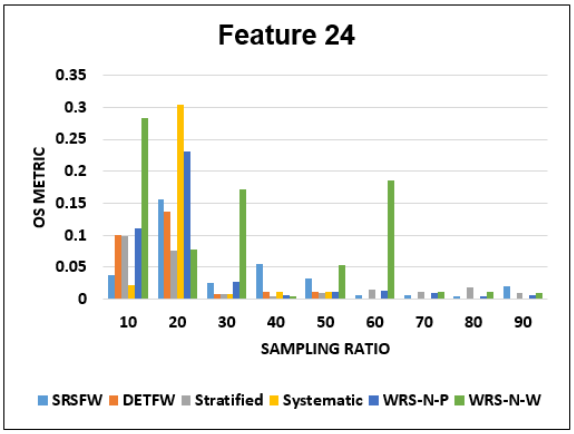
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3118605, IEEE Access



(a) Standard deviation and median estimation



(b) Mean estimation



(c) OS metric.

FIGURE 21: Statistical metrics estimation of feature 24 using non-stream sampling policies.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3118605, IEEE Access

Figures 22, 23, and 24, and Tables 5, 6, and 7 show that for all the stream algorithms, except the DETSW the Chain-sample algorithms, the window size has no considerable impact on the OS value of features 5, 7, and 8. changing the window size does not change the OS value. This can be explained by the fact that the elements are selected in a deterministic manner. Regarding the Chain-sample algorithm, the OS value increases when the value of the window increases because of the collision problem. Results also show that the OS value of the priority sampling algorithm is almost zero when the sampling rate is $\in [60, 90]$. Results also show that the OS value of the DETSW algorithm is stable and very close to 0 when the sampling ratio is $\in [60\%, 90\%]$.

Results in Figure 22 show that while for all the stream algorithms the OS value reaches almost its minimum when the sampling rate is equal 90%, the OS value's variation according to the sampling ratio is dependent on the sampling policy. For instance, for the SRSSW, Chain+, Reservoir, and RP sampling algorithms, the minimum OS value is achieved when the sampling rate is equal to 80%. For the Backing algorithms, it is achieved for a sampling rate equal to 60%. For the StreamSamp algorithm, it is achieved for a sampling rate equal to 70%.

Results in Figure 23 show that while for all the stream algorithms the OS value reaches almost its minimum when the sampling rate is equal 90%, the OS value's variation according to the sampling ratio is dependent on the sampling policy. For instance, for the SRSSW, Chain+, and RP sampling algorithms, the minimum OS value is achieved when the sampling rate is equal to 80%. For the Reservoir algorithm, it is achieved for a sampling rate equal to 60%. For the Backing algorithm, it is achieved for a sampling rate equal to 20%. For the StreamSamp algorithm, it is achieved for a sampling rate equal to 30%.

Results in Figure 24 show that while for all the stream algorithms the OS value reaches almost its minimum when the sampling rate is equal 90%, the OS value's variation according to the sampling ratio is dependent on the sampling policy. For instance, for the SRSSW and Chain+ sampling algorithms, the minimum OS value is achieved when the sampling rate is equal to 70%. For the Reservoir and StreamSamp algorithms, it is achieved for a sampling rate equal to 80%. For the Backing algorithm, it is achieved for a sampling rate equal to 30%. For the RP algorithm, it is achieved for a sampling rate equal to 60%.

As a conclusion, Table 8 shows the stream sampling policies and the corresponding sampling rates ($\in [10\%, 80\%]$) that can be used to achieve low OS values for features 5, 7, and 8.

2) Probe attack

Figures 22, 25, 26, 27, and Tables 5, 9, 10, and 11 show that for all the stream algorithms, except the DETSW the Chain-sample algorithms, the window size has no considerable impact on the OS value for features 5, 28, 30, and 36. Regarding the DETSW, the OS value remains the same when the window size changes. This can be explained by the fact that the elements are selected in a deterministic manner. Regarding the Chain-sample algorithm, the OS value increases when the value of the window increases. Results also show that the OS value of the priority sampling algorithm is almost zero when the sampling rate is $\in [60, 90]$.

Results in Figures 22, 25, 26, and 27 show that the OS value of the priority sampling algorithm is almost zero when the sampling rate is $\in [60, 90]$. Results also show that the OS value of the DETSW and priority algorithms is stable when the sampling ratio is $\in [60\%, 90\%]$.

Results in Figure 25 show that while for all the algorithms the OS value reaches almost its minimum when the sampling rate is equal 90%, the OS value's variation according to the sampling ratio is dependent on the sampling policy. For instance, for the SRSSW and Chain+ sampling algorithms, the minimum OS value is achieved when the sampling rate is equal to 70%. For the Reservoir and RP algorithms, it is achieved for a sampling rate equal to 80%. For the Backing and StreamSamp algorithms, it is achieved for a sampling rate equal to 40%.

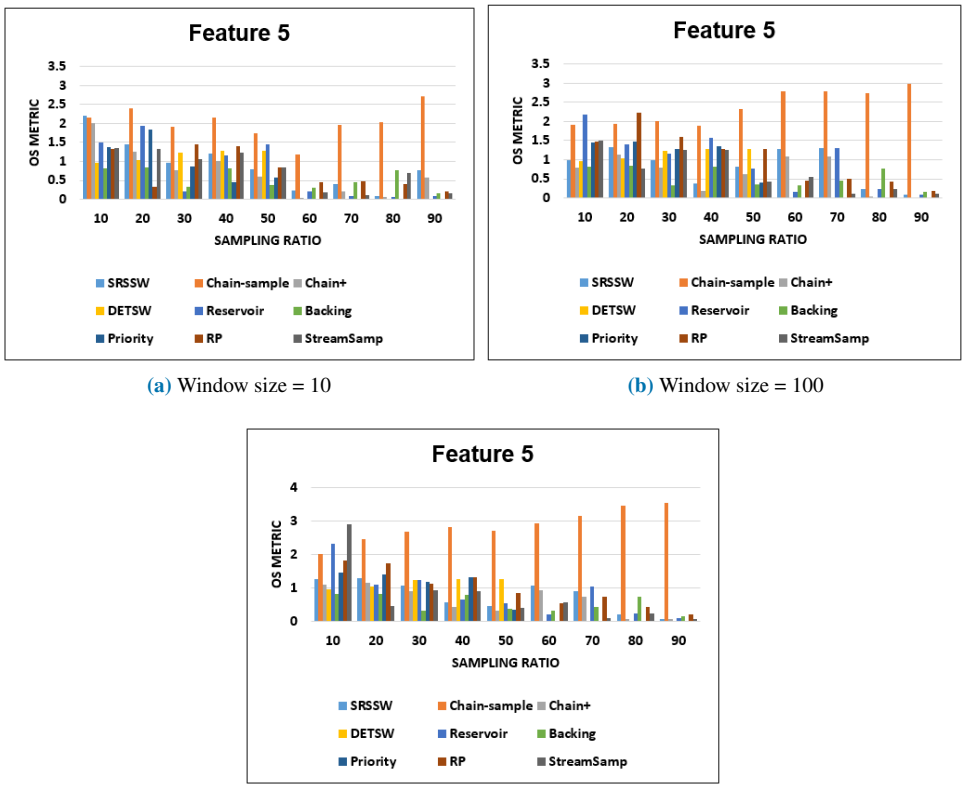
Results in Figure 26 show that while for all the algorithms the OS value reaches almost its minimum when the sampling rate is equal 90%, the OS value's variation according to the sampling ratio is dependent on the sampling policy. For instance, for the SRSSW, Chain+, and StreamSamp sampling algorithms, the minimum OS value is achieved when the sampling rate is equal to 80%. For the Reservoir algorithm, it is achieved for a sampling rate equal to 40%. For the Backing algorithm, it is achieved for a sampling rate equal to 60%. For the RP it is achieved for a sampling rate equal to 70%.

Results in Figure 27 show that while for all the algorithms the OS value reaches almost its minimum when the sampling rate is equal 90%, the OS value's variation according to the sampling ratio is dependent on the sampling policy. For instance, for the SRSSW and Chain+ sampling algorithms, the minimum OS value is achieved when the sampling rate is equal to 70%. For the Reservoir algorithm, it is achieved for a sampling rate equal to 80%. For the Backing and StreamSamp algorithm, it is achieved for a sampling rate equal to 50%. For the RP it is achieved for a sampling rate equal to 40%.

As a conclusion, Table 12 shows the stream sampling policies and the corresponding sampling rates ($\in [10\%, 80\%]$) that can be used to achieve low OS values for features 5, 28, 30, and 36.

3) R2L attack

Figures 28, 29, 30, and 31, and Tables 13, 14, 15, and 16 show that for all the stream algorithms, except the DETSW the Chain-sample algorithms, the window size has no considerable impact on the OS value for features 6, 11, 12, and 39. Regarding the DETSW, the OS value remains the same when the window size changes. This can be explained by the fact that the elements are selected in a deterministic manner. Regarding the Chain-sample algorithm, the OS value increases



(a) Window size = 10

(b) Window size = 100

(c) Window size = 1000

FIGURE 22: Statistical metrics estimation of feature 5 using stream sampling policies.

TABLE 5: Variation of the OS value of feature 5 for stream algorithms for different window sizes.

	SRSSW	Chain	Chain+	DETSW	Reservoir	Backing	Priority	RP	StreamSamp
n = 10	0.899	2.027	0.720	0.643	0.741	0.540	0.568	0.766	0.772
n = 100	0.826	2.376	0.626	0.643	0.982	0.540	0.665	1.045	0.693
n = 1000	0.773	2.865	0.638	0.643	0.825	0.534	0.634	0.976	0.730
OS Average	0.833	2.422	0.661	0.643	0.849	**0.538**	0.622	0.929	0.732

TABLE 6: Variation of the OS value of feature 7 for stream algorithms for different window sizes.

	SRSSW	Chain	Chain+	DETSW	Reservoir	Backing	Priority	RP	StreamSamp
n = 10	0.382	1.530	0.332	0.298	0.674	1.117	0.210911657	0.629	0.607
n = 100	0.687	2.552	0.657	0.298	0.929	1.117	0.280	0.925	0.388
n = 1000	0.866	3.381	0.826	0.298	0.710	1.117	0.437	0.757	0.598
OS Average	0.645	2.488	0.605	**0.298**	0.771	1.117	0.309	0.770	0.531

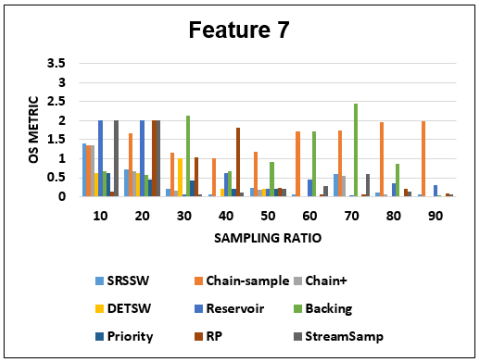
TABLE 7: Variation of the OS value of feature 8 for stream algorithms for different window sizes.

	SRSSW	Chain	Chain+	DETSW	Reservoir	Backing	Priority	RP	StreamSamp
n = 10	0.201	0.568	0.181	0.122	0.207	0.406	0.138	0.269	0.239
n = 100	0.260	0.606	0.240	0.122	0.194	0.4184	0.073	0.181	0.133
n = 1000	0.193	0.432	0.183	0.122	0.146	0.404	0.138	0.111	0.185
OS Average	0.218	0.535	0.202	0.122	0.182	0.409	**0.116**	0.187	0.186

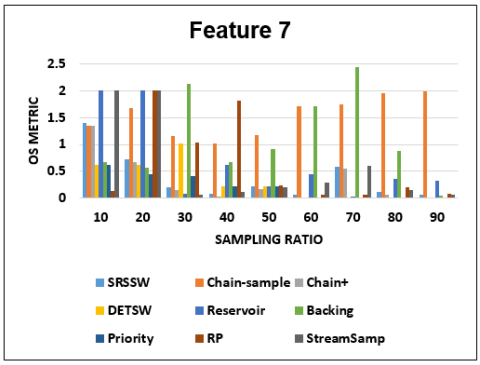
TABLE 8: Lowest achieved OS values according to the stream policies and sampling rates, for DoS attack features.

Feature	Sampling Policy								
	SRSSW	Chain	Chain+	DETSW	Reservoir	Backing	Priority	RP	StreamSamp
5	**80%**	**10%**	**80%**	**60%**	**80%**	**60%**	**60%**	**80%**	**70%**
	0.137	1.245	0.067	0.0004	0.171	0.318	**0**	0.421	**0.104**
7	**80%**	**10%**	**80%**	**60%**	**60%**	**20%**	**60%**	**80%**	**30%**
	0.197	2.06	0.161	0.0002	0.264	0.574	**0**	0.446662	0.070885
8	**70%**	**10%**	**70%**	**60%**	**80%**	**30%**	**60%**	**60%**	**80%**
	0.047	0.455	0.028	**0.0006**	0.076	0.101	**0**	0.068	0.048

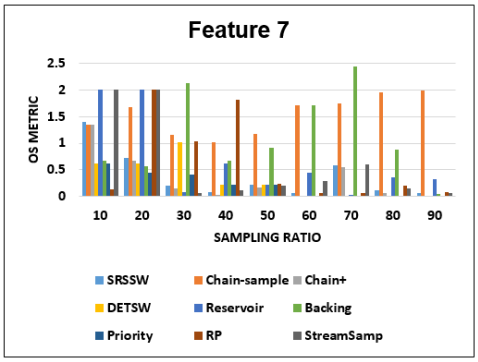
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3118605, IEEE Access



(a) Window size = 10

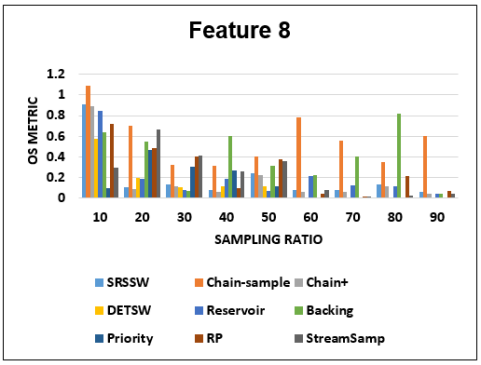


(b) Window size = 100

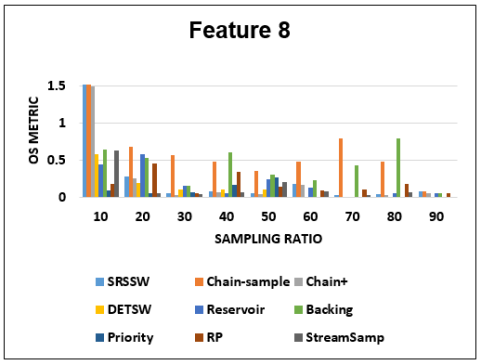


(c) Window size = 1000

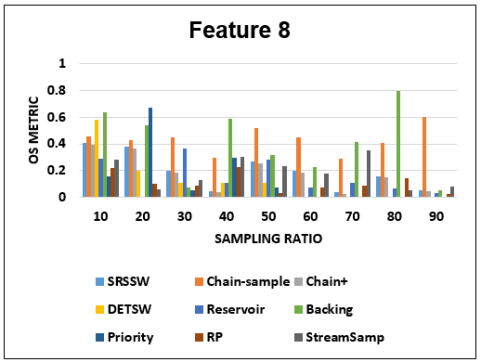
FIGURE 23: Statistical metrics estimation of feature 7 using stream sampling policies.



(a) Window size = 10



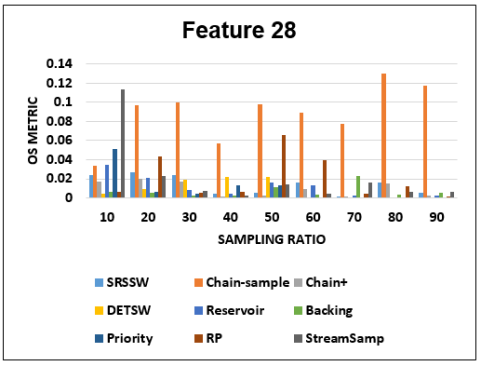
(b) Window size = 100



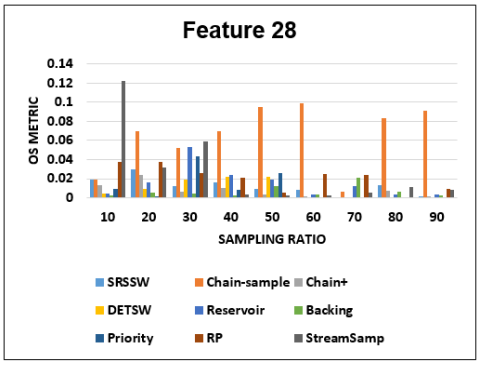
(c) Window size = 1000

FIGURE 24: Statistical metrics estimation of feature 8 using stream sampling policies.

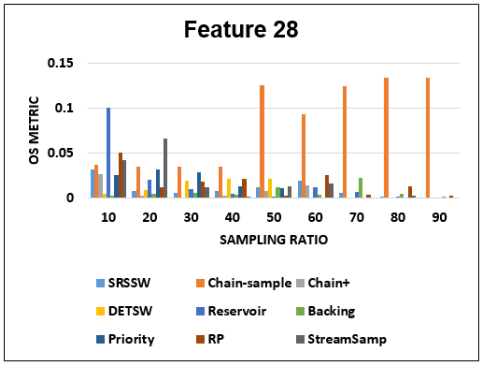
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3118605, IEEE Access



(a) Window size = 10



(b) Window size = 100



(c) Window size = 1000

FIGURE 25: Statistical metrics estimation of feature 28 using stream sampling policies.

TABLE 9: Variation of the OS value of feature 28 for stream algorithms for different window sizes.

	SRSSW	Chain-sample	Chain+	DETSW	Reservoir	Backing	Priority	RP	StreamSamp
n = 10	0.013	0.088	0.009	0.008	0.011	0.006	0.009	0.020	0.021
n = 100	0.0122	0.065	0.007	0.008	0.0154	0.006	0.009	0.020	0.027
n = 1000	0.010	0.083	0.006	0.008	0.017	0.006	0.012	0.016	0.017
OS Average	0.011	0.079	0.007	0.008	0.014	**0.006**	0.010	0.019	0.022

TABLE 10: Variation of the OS value of feature 30 for stream algorithms for different window sizes.

	SRSSW	Chain-sample	Chain+	DETSW	Reservoir	Backing	Priority	RP	StreamSamp
n = 10	0.014	0.275	0.012	0.020	0.020	0.030	0.020	0.024	0.020
n = 100	0.036	0.497	0.029	0.020	0.020	0.029	0.014	0.031	0.013
n = 1000	0.029	2.829	0.026	0.020	0.028	0.030	0.019	0.028	0.038
OS Average	0.026	1.200	0.023	0.020	0.022	0.030	**0.018**	0.027	0.024

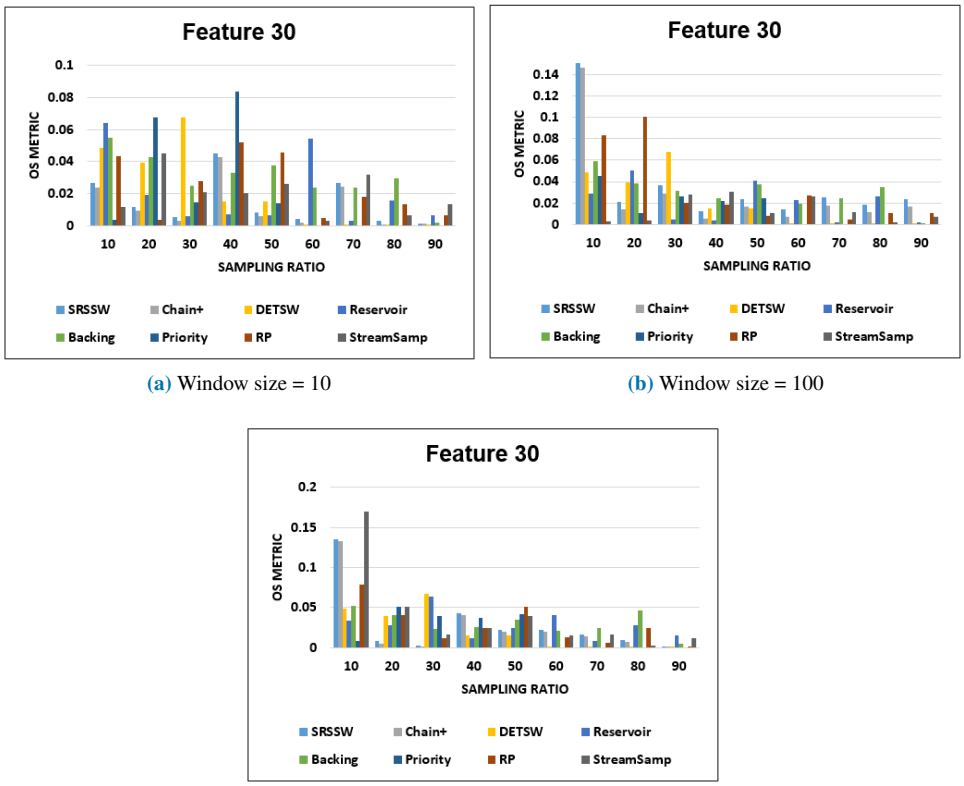
TABLE 11: Variation of the OS value of feature 36 for stream algorithms for different window sizes.

	SRSSW	Chain-sample	Chain+	DETSW	Reservoir	Backing	Priority	RP	StreamSamp
n = 10	0.021	0.891	0.018	0.015	0.0177	0.067	0.018	0.023	0.023
n = 100	0.025	1.475	0.015	0.015	0.024	0.069	0.021	0.025	0.022
n = 1000	0.021	2.471	0.015	0.015	0.029	0.069	0.016	0.017	0.025
OS Average	0.022	1.612	0.016	0.015	0.024	0.068	**0.019**	0.022	0.023

TABLE 12: Lowest achieved OS values according to the stream policies and sampling rates, for Probe attack features.

Feature	Sampling Policy								
	SRSSW	Chain-sample	Chain+	DETSW	Reservoir	Backing	Priority	RP	StreamSamp
5	**80%**	**10%**	**80%**	**60%**	**80%**	**60%**	**60%**	**80%**	**70%**
	0.137	1.245	0.067	**0.0004**	0.171	0.318	**0**	0.421	0.003
28	**70%**	**10%**	**70%**	**60%**	**80%**	**40%**	**60%**	**80%**	**40%**
	0.002	0.037	0.002	**0.0001**	0.001	0.002	**0**	0.008	0.002
30	**80%**	**10%**	**80%**	**60%**	**40%**	**60%**	**60%**	**70%**	**80%**
	0.01043	2.935	0.014	**0.0001**	0.007	0.02127	**0**	0.009	0.003
36	**70%**	**10%**	**70%**	**60%**	**80%**	**50%**	**60%**	**40%**	**50%**
	0.007	2.463	0.014	**0.00004**	0.003	0.0193	**0**	0.003	0.008

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3118605, IEEE Access

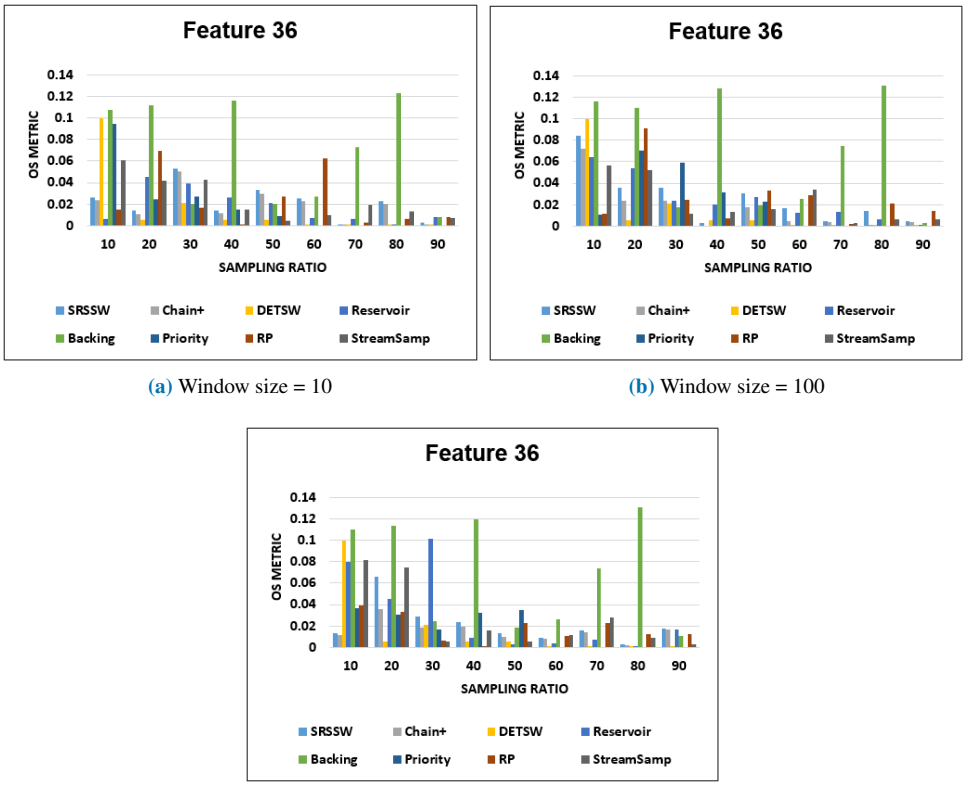


(a) Window size = 10

(b) Window size = 100

(c) Window size = 1000

FIGURE 26: Statistical metrics estimation of feature 30 using stream sampling policies.



(a) Window size = 10

(b) Window size = 100

(c) Window size = 1000

FIGURE 27: Statistical metrics estimation of feature 36 using stream sampling policies.

when the value of the window increases. Results also show that the OS value of the priority sampling algorithm is almost zero when the sampling rate is $\in [60, 90]$. Results also show that the backing sampling algorithm presents the highest OS value whatever is the sampling ratio.

Results in Figures 28, 29, 30, and 31 show that the OS value of the priority sampling algorithm is almost zero when the sampling rate is $\in [60, 90]$. Results also show that the OS value of the DETSW and priority algorithms is stable when the sampling ratio is $\in [60\%, 90\%]$.

Results in Figure 28 show that while for all the algorithms the OS value reaches almost its minimum when the sampling rate is equal 90%, the OS value's variation according to the sampling ratio is dependent on the sampling policy. For instance, for the SRSSW and Chain+ sampling algorithms, the minimum OS value is achieved when the sampling rate is equal to 70%. For the Reservoir and Backing algorithms, it is achieved for a sampling rate equal to 60%. For the RP and StreamSamp algorithms, it is achieved for a sampling rate equal to 80%.

Results in Figure 29 show that while for all the algorithms the OS value reaches almost its minimum when the sampling rate is equal 90%, the OS value's variation according to the sampling ratio is dependent on the sampling policy. For instance, for the SRSSW, Chain+, and Backing sampling algorithms, the minimum OS value is achieved when the sampling rate is equal to 60%. For Reservoir sampling, it is achieved for a sampling rate equal to 80%. For the RP and StreamSamp sampling algorithms, it is achieved for a sampling rate equal to 70%.

Results in Figure 30 show that while for all the algorithms the OS value reaches almost its minimum when the sampling rate is equal 90%, the OS value's variation according to the sampling ratio is dependent on the sampling policy. For instance, for the SRSSW, Chain+, and Reservoir sampling algorithms, the minimum OS value is achieved when the sampling rate is equal to 80%. For the Backing sampling, it is achieved for a sampling rate equal to 50%, for the RP sampling, it is achieved for a sampling rate equal to 60%, for the StreamSamp sampling, it is achieved for a sampling rate equal to 70%.

Results in Figure 31 show that while for all the algorithms the OS value reaches almost its minimum when the sampling rate is equal 90%, the OS value's variation according to the sampling ratio is dependent on the sampling policy. For instance, for the SRSSW, Chain+, and StreamSamp sampling algorithms, the minimum OS value is achieved when the sampling rate is equal to 70%. For Reservoir and RP sampling algorithms, it is achieved for a sampling rate equal to 80%. For the Backing sampling, it is achieved for a sampling rate equal to 30%.

As a conclusion, Table 17 shows the stream sampling policies and the corresponding sampling rates ($\in [10\%, 80\%]$) that can be used to achieve low OS values for features 6, 11, 23, and 39.

4) U2R attack

Figures 32, 33, 27, and Tables 18, 19, and 11 show that for all the stream algorithms, except the DETSW the Chain-sample algorithms, the window size has no considerable impact on the OS value for features 14, 24, and 36. Regarding the DETSW, the OS value remains the same when the window size changes. This can be explained by the fact that the elements are selected in a deterministic manner. Regarding the Chain-sample algorithm, the OS value increases when the value of the window increases. Results also show that the OS value of the priority sampling algorithm is almost zero when the sampling rate is $\in [60, 90]$. Results also show that the backing sampling algorithm presents the highest OS value whatever is the sampling ratio.

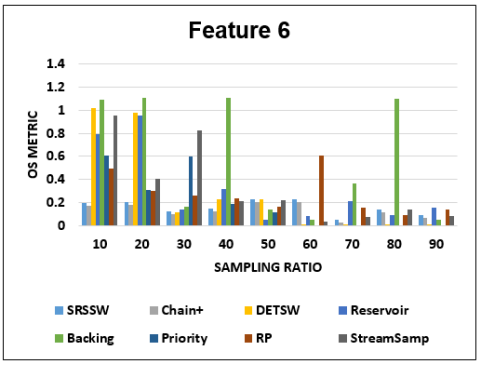
Results in Figures 32, 33, and 27 show that the OS value of the priority sampling algorithm is almost zero when the sampling rate is $\in [60, 90]$. Results also show that the OS value of the DETSW and priority algorithms is stable when the sampling ratio is $\in [60\%, 90\%]$.

Results in Figure 32 show that while for all the algorithms the OS value reaches almost its minimum when the sampling rate is equal 90%, the OS value variation according to the sampling ratio is dependent on the sampling policy. For instance, for the SRSSW, Chain+, and RP sampling algorithms, the minimum OS value is achieved when the sampling rate is equal to 80%. For the StreamSamp algorithm, it is achieved for a sampling rate equal to 50%. For the Reservoir and Backing sampling, it is achieved for a sampling rate equal to 70%.

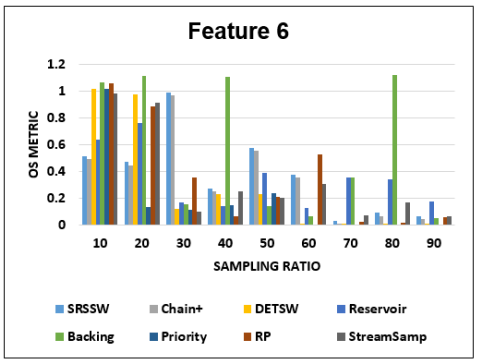
Results in Figure 33 show that while for all the algorithms the OS value reaches almost its minimum when the sampling rate is equal 90%, the OS value variation according to the sampling ratio is dependent on the sampling policy. For instance, for the SRSSW, Chain+, and RP sampling algorithms, the minimum OS value is achieved when the sampling rate is equal to 70%. For the Reservoir and StreamSamp algorithm, it is achieved for a sampling rate equal to 80%. For the Backing, it is achieved for a sampling rate equal to 50%.

As a conclusion, Table 20 shows the stream sampling policies and the corresponding sampling rates ($\in [10\%, 80\%]$) that can be used to achieve low OS values for features 14, 24, and 36.

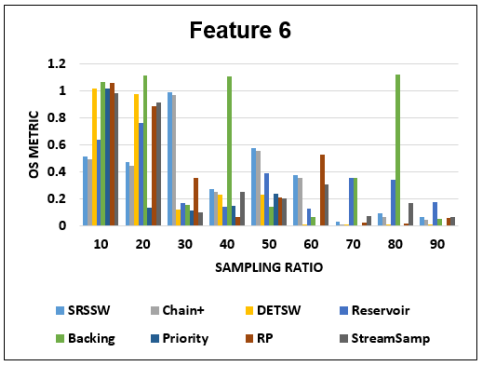
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3118605, IEEE Access



(a) Window size = 10



(b) Window size = 100



(c) Window size = 1000

FIGURE 28: Statistical metrics estimation of feature 6 using stream sampling policies.

TABLE 13: Variation of the OS value of feature 6 for stream algorithms for different window sizes.

	SRSSW	Chain-sample	Chain+	DETSW	Reservoir	Backing	Priority	RP	StreamSamp
n = 10	0.155	1.624	0.132	0.286	0.311	0.574	0.202	0.273	0.327
n = 100	0.376	2.698	0.353	0.286	0.343	0.575	0.182	0.356	0.340
n = 1000	0.263	3.568	0.240	0.286	0.277	0.572	0.182	0.244	0.354
OS Average	0.265	2.630	0.242	0.286	0.310	0.574	**0.189**	0.291	0.341

TABLE 14: Variation of the OS value of feature 11 for stream algorithms for different window sizes.

	SRSSW	Chain+	DETSW	Reservoir	Backing	Priority	RP	StreamSamp
n = 10	0.065	0.047	0.049	0.098	0.173	0.031	0.099	0.098
n = 100	0.037	0.021	0.049	0.058	0.185	0.066	0.100	0.061
n = 1000	0.074	0.056	0.049	0.074	0.187	0.066	0.076	0.040
OS Average	0.059	**0.041**	0.049	0.077	0.182	0.054	0.092	0.066

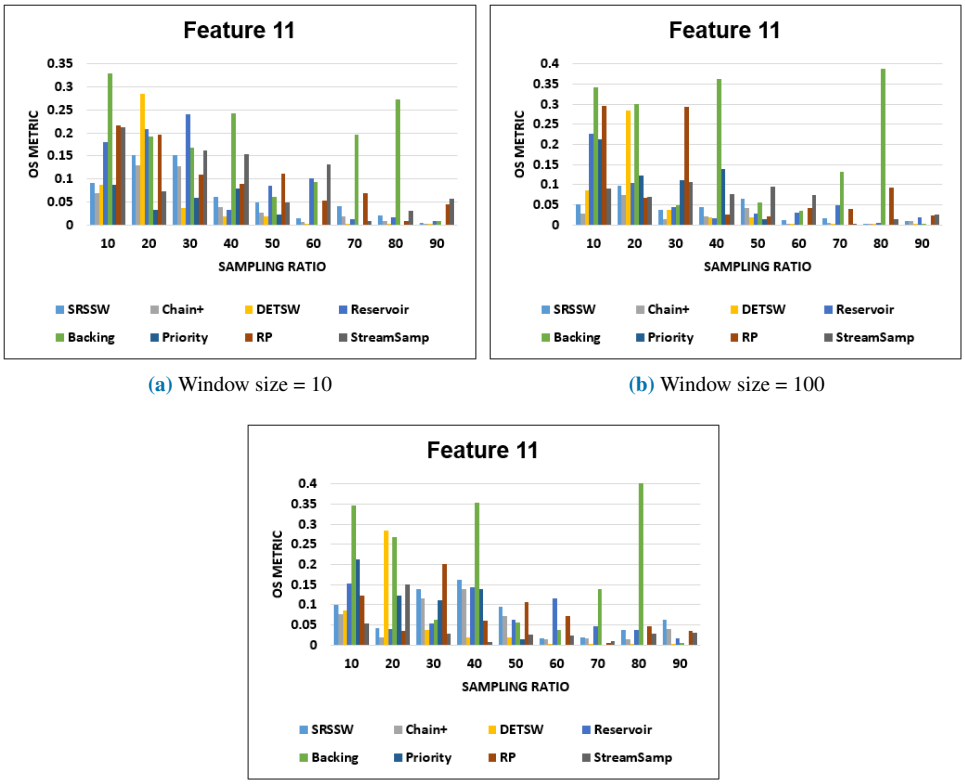
TABLE 15: Variation of the OS value of feature 23 for stream algorithms for different window sizes.

	SRSSW	Chain+	DETSW	Reservoir	Backing	Priority	RP	StreamSamp
n = 10	0.012	0.008	0.015	0.020	0.108	0.018	0.012	0.018
n = 100	0.032	0.028	0.015	0.050	0.092	0.038	0.069	0.031
n = 1000	0.0207	0.019	0.015	0.030	0.107	0.038	0.015	0.035
OS Average	0.021	0.018	**0.015**	0.0335	0.102	0.0315	0.0327	0.0284

TABLE 16: Variation of the OS value of feature 39 for stream algorithms for different window sizes.

	SRSSW	Chain+	DETSW	Reservoir	Backing	Priority	RP	StreamSamp
n = 10	0.022	0.018	0.019	0.033	0.027	0.027	0.018	0.019
n = 100	0.018	0.017	0.019	0.029	0.027	0.018	0.022	0.029
n = 1000	0.021	0.019	0.019	0.013	0.032	0.018	0.027	0.020
OS Average	0.020	0.0187	**0.019**	0.025	0.028	0.021	0.022	0.023

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3118605, IEEE Access

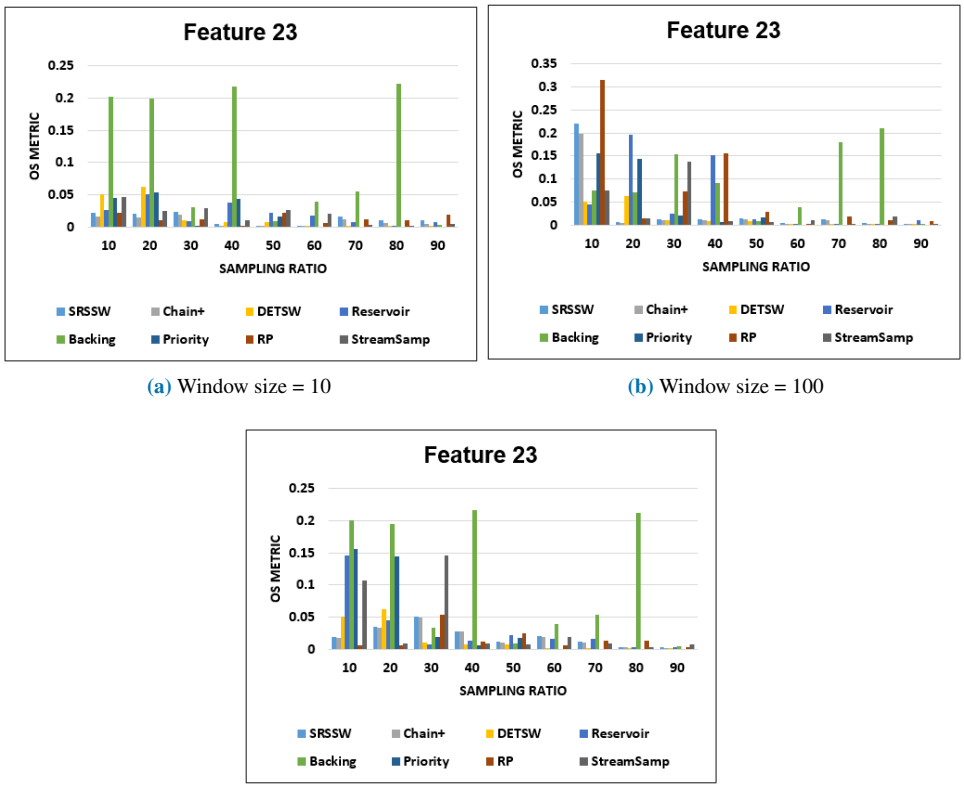


(a) Window size = 10

(b) Window size = 100

(c) Window size = 1000

FIGURE 29: Statistical metrics estimation of feature 11 using stream sampling policies.



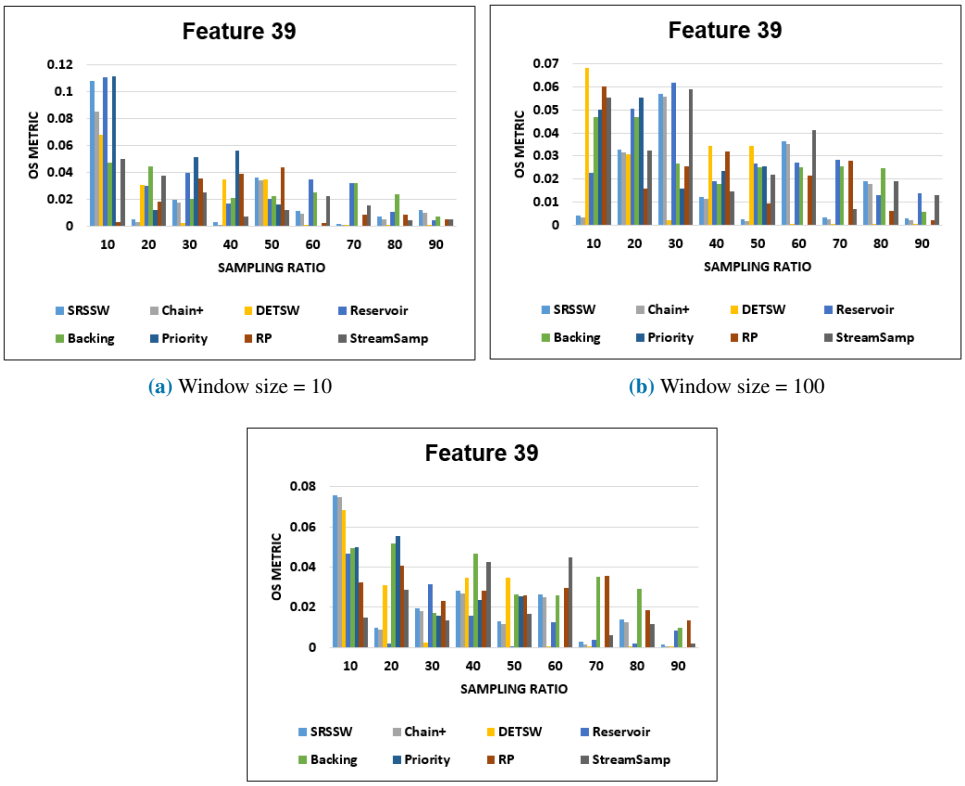
(a) Window size = 10

(b) Window size = 100

(c) Window size = 1000

FIGURE 30: Statistical metrics estimation of feature 23 using stream sampling policies.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3118605, IEEE Access



(a) Window size = 10

(b) Window size = 100

(c) Window size = 1000

FIGURE 31: Statistical metrics estimation of feature 39 using stream sampling policies.

TABLE 17: Lowest achieved OS values according to the stream policies and sampling rates, for R2L attack features.

Feature	Sampling Policy								
	SRSSW	**Chain-sample**	**Chain+**	**DETSW**	**Reservoir**	**Backing**	**Priority**	**RP**	**StreamSamp**
6	**70%**	**10%**	**70%**	**60%**	**60%**	**60%**	**60%**	**80%**	**80%**
	0.066	3.092	0.097	**0.0003**	0.099	0.058	**0**	0.105	0.109
11	**60%**	**10%**	**60%**	**60%**	**80%**	**60%**	**60%**	**70%**	**70%**
	0.014	2.198	0.014	**0.0006**	0.0199	0.055	**0**	0.039	0.007
23	**80%**	**10%**	**80%**	**60%**	**80%**	**50%**	**60%**	**60%**	**70%**
	0.006	2.719	**0.002**	0.0003	**0.002**	0.009	**0**	0.004	0.005
39	**70%**	**10%**	**70%**	**60%**	**80%**	**30%**	**60%**	**80%**	**70%**
	0.002	2.525	0.0015	**0.0006**	0.008	0.021	**0**	0.0111	0.009

TABLE 18: Variation of the OS value of feature 14 for stream algorithms for different window sizes.

	SRSSW	**Chain+**	**DETSW**	**Reservoir**	**Backing**	**Priority**	**RP**	**StreamSamp**
n = 10	0.136	0.113	0.199	0.084	0.273	0.153	0.293	0.154
n = 100	0.253	0.109	0.199	0.183	0.243	0.166	0.389	0.074
n = 1000	0.159	0.150	0.199	0.194	0.266	0.064	0.178	0.145
OS Average	0.183	**0.124**	0.199	0.154	0.261	0.128	0.287	**0.124**

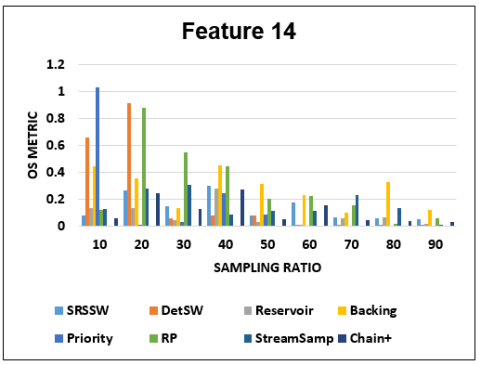
TABLE 19: Variation of the OS value of feature 24 for stream algorithms for different window sizes.

	SRSSW	**Chain+**	**DETSW**	**Reservoir**	**Backing**	**Priority**	**RP**	**StreamSamp**
n = 10	0.075	0.064	0.015	0.081	0.182	0.059	0.100	0.073
n = 100	0.092	0.078	0.015	0.048	0.164	0.042	0.026	0.060
n = 1000	0.050	0.046	0.015	0.032	0.186	0.085	0.02807163	0.042
OS Average	0.072	0.062	**0.015**	0.054	0.177	0.062	0.051	0.058

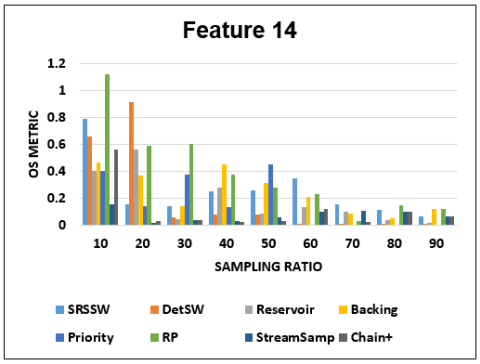
TABLE 20: Lowest achieved OS values according to the stream policies and sampling rates, for U2R attack features.

Feature	Sampling Policy								
	SRSSW	**Chain-sample**	**Chain+**	**DETSW**	**Reservoir**	**Backing**	**Priority**	**RP**	**StreamSamp**
14	**80%**	**10%**	**80%**	**60%**	**70%**	**70%**	**60%**	**80%**	**50%**
	0.075	2.112	0.049	**0.0006**	0.071	0.093	**0**	0.069	0.088
24	**70%**	**10%**	**70%**	**60%**	**80%**	**50%**	**60%**	**70%**	**80%**
	0.006	2.104	0.001	**0.0004**	0.011714	0.059	**0**	0.010	0.005
36	**70%**	**10%**	**70%**	**60%**	**80%**	**50%**	**60%**	**40%**	**50%**
	0.007	2.463	0.014	**0.00004**	0.003	0.0193	**0**	0.0030	0.008

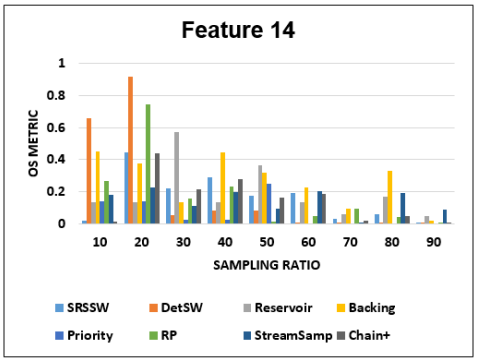
This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3118605, IEEE Access



(a) Window size = 10

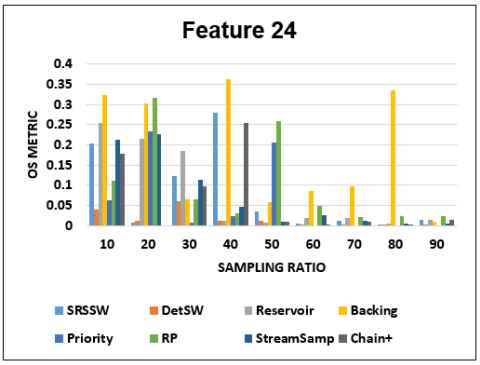


(b) Window size = 100

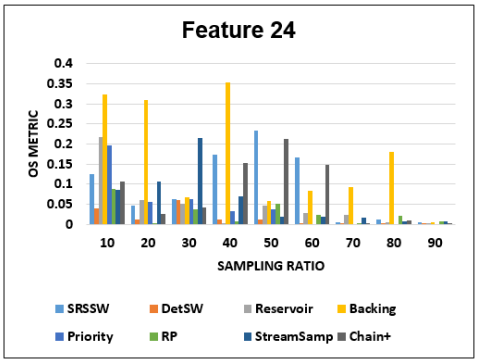


(c) Window size = 1000

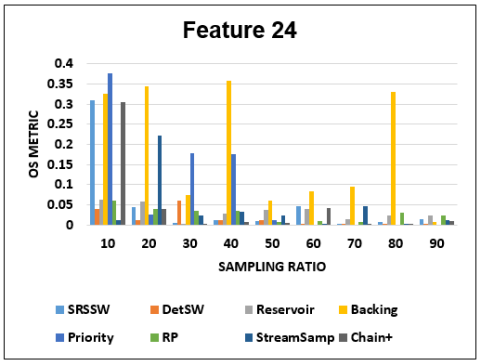
FIGURE 32: Statistical metrics estimation of feature 14 using stream sampling policies.



(a) Window size = 10



(b) Window size = 100



(c) Window size = 1000

FIGURE 33: Statistical metrics estimation of feature 24 using stream sampling policies.

V. CONCLUSION AND OPEN RESEARCH CHALLENGES

In this paper, we investigated the statistical impact of network traffic sampling to quantify the amount of deterioration that the sampling process introduces with respect to non-sampled traffic. By performing an off-line analysis of the NSL-KDD dataset, we carried out an experimental comparison of existing sampling techniques and studied their impact on several well-known statistical measures to assess the level of degradation introduced by sampling. Different sampling policies were evaluated, and different features and attacks were considered.

Our study suffers from the following limitations:

- Without loss of generality, in our work, we evaluated the performance of sampling algorithms on intrusion detection using the NSl-KDD database. However, there are other data sets, such as CAIDA [41], CIDDS [42], etc. In our future work, we aim to consider data from other modern networks or even real networks.
- Features preprocessing and feature selection process should be conducted in advance of intrusion detection. Since the network traffic is enormous, analysis and intrusion detection become difficult. On the other hand, there could be a relation between the network characteristics. Some of these features may even be redundant or irrelevant. Thus, it is necessary to reduce the volume of traffic data to be processed and analyzed through a feature selection process. This process identifies the relevant characteristics of the traffic that leads to improved performance of the IDS. In this work, we referred to many recent studies to determine the appropriate features of each form of attack. We plan in our future work to study different feature selection algorithms to determine the most precise one, and therefore, to select the most relevant features for each attack.
- The NSL-KDD database contains about $150K$ records divided into training and testing subsets. It consists of 41 attributes and includes 22 attacks types. These attacks are of four categories: DoS, probe, R2L, and U2R. In our future work, we aim to test the impact of sampling on the detection of many other types of network attacks.

Knowing the traffic parameters, a convenient sampling algorithm with calculated compromise (accuracy vs. computational cost) can be configured and fine-tuned accordingly. Several open issues that would benefit from further studies can be identified.

- *Static vs. dynamic packets sampling*. The sampling ratio impacts directly the accuracy of the built sample [43], [44]. Static packets' sampling algorithms have been used for many years. With these algorithms, all items are somehow selected randomly, except for the deterministic sampling, with a predefined sampling ratio. Nevertheless, given the dynamic nature of network traffic, static sampling cannot guarantee the estimation accuracy and is, thus, poorly suited for network monitoring. During periods of idle activity or low network loads, a long sampling interval provides sufficient accuracy at minimal overhead. However, bursts of high activity require shorter sampling intervals to accurately measure network status at the expense of increased sampling overhead. To preserve the accuracy and provide accurate estimations, the sampling policy should adapt to the network state. It is worth noting that network devices have certain limits in terms of resources available for sampling. Some network devices might even stop sampling during traffic bursts. To address these issues, adaptive sampling algorithms can be designed and applied to dynamically adjust the sampling interval and optimize the sampling and traffic classification accuracy. Dynamic sampling algorithms have dynamic sampling rates, this allows them to control the accuracy of the sample by controlling the number of measurements to be sampled. A decision here should be taken in advance to adjust the sampling ratio, before network traffic change.
- *On-the-fly learning*. High-speed network traffic is dynamic and volatile thus a responsive packets sampling algorithm is vital for robust and timely anomaly detection. For a sampling algorithm to be responsive, fast traffic features learning is a prerequisite. Fast and accurate on-the-fly features learning is an open challenge to be studied especially with the adequacy of data mining (such as time series) and artificial intelligence algorithms for this task. In this context, various prediction and forecasting techniques could be used to predict network traffic and any potential change in its characteristics.
- *Weighted Sampling Algorithm*. As the benchmarking results showed, not all features are equal in predicting anomalies. Some features are more sensitive to change thus can be used as an early warning for potential anomalies. Additionally, not all packets are equal. Designing a multi-feature weighted sampling algorithm can benefit from sensitivity and accuracy if successfully configured.
- *Data Quality*. The arriving packets can be contaminated (delayed, distorted, etc.) or even lost before reaching the IDS. This is very frequent in the case of network congestion, noisy channels, and unstable changes to the network topology. The missing data can be very random and sporadic, resulting in very distorted measurements by the IDS, following the survivor bias. Studying the impact of missing data or more generally data quality is also an open issue.

REFERENCES

[1] Mohamed Faisal Elrawy, Ali Ismail Awad, and Hesham FA Hamed. Intrusion detection systems for iot-based smart environments: a survey. Journal of Cloud Computing, 7(1):1–20, 2018.

[2] Nathan Tuck, Timothy Sherwood, Brad Calder, and George Varghese. Deterministic memory-efficient string matching algorithms for intrusion detection. In IEEE INFOCOM 2004, volume 4, pages 2628–2639. IEEE, 2004.

[3] Mohammad Masdari and Marzie Jalali. A survey and taxonomy of dos

attacks in cloud computing. Security and Communication Networks, 9(16):3724–3751, 2016.

[4] Giampaolo Bovenzi, Giuseppe Aceto, Domenico Ciuonzo, Valerio Persico, and Antonio Pescapé. A hierarchical hybrid intrusion detection approach in iot scenarios. In GLOBECOM 2020-2020 IEEE Global Communications Conference, pages 1–7. IEEE, 2020.

[5] Ankit Thakkar and Ritika Lohiya. A survey on intrusion detection system: feature selection, model, performance measures, application perspective, challenges, and future research directions. Artificial Intelligence Review, pages 1–111, 2021.

[6] Shigang Chen and Klara Nahrstedt. An overview of quality of service routing for next-generation high-speed networks: problems and solutions. IEEE network, 12(6):64–79, 1998.

[7] Michele Colajanni and Mirco Marchetti. A parallel architecture for stateful intrusion detection in high traffic networks. In Proc. of the IEEE/IST Workshop on" Monitoring, attack detection and mitigation"(MonAM 2006), Tuebingen, Germany, 2006.

[8] Matthias Vallentin, Robin Sommer, Jason Lee, Craig Leres, Vern Paxson, and Brian Tierney. The nids cluster: Scalable, stateful network intrusion detection on commodity hardware. In International Workshop on Recent Advances in Intrusion Detection, pages 107–126. Springer, 2007.

[9] Giuseppe Aceto, Domenico Ciuonzo, Antonio Montieri, and Antonio Pescapé. Mobile encrypted traffic classification using deep learning: Experimental evaluation, lessons learned, and challenges. IEEE Transactions on Network and Service Management, 16(2):445–458, 2019.

[10] Paul D Amer and Lillian N Cassel. Management of sampled real-time network measurements. In [1989] Proceedings. 14th Conference on Local Computer Networks, pages 62–63. IEEE Computer Society, 1989.

[11] Daniela Brauckhoff, Bernhard Tellenbach, Arno Wagner, Martin May, and Anukool Lakhina. Impact of packet sampling on anomaly detection metrics. In Proceedings of the 6th ACM SIGCOMM conference on Internet measurement, pages 159–164. ACM, 2006.

[12] Jianning Mai, Ashwin Sridharan, Chen-Nee Chuah, Hui Zang, and Tao Ye. Impact of packet sampling on portscan detection. IEEE Journal on Selected Areas in Communications, 24(12):2285–2298, 2006.

[13] Jianning Mai, Chen-Nee Chuah, Ashwin Sridharan, Tao Ye, and Hui Zang. Is sampled data sufficient for anomaly detection? In Proceedings of the 6th ACM SIGCOMM conference on Internet measurement, pages 165–176. ACM, 2006.

[14] Antonio Pescapé, Dario Rossi, Davide Tammaro, and Silvio Valenti. On the impact of sampling on traffic monitoring and analysis. In 2010 22nd International Teletraffic Congress (lTC 22), pages 1–8. IEEE, 2010.

[15] Hu Zhang, Jun Liu, Wenli Zhou, and Shuo Zhang. Sampling method in traffic logs analyzing. In Intelligent Human-Machine Systems and Cybernetics (IHMSC), 2016 8th International Conference on, volume 1, pages 554–558. IEEE, 2016.

[16] Gilles Roudière and Philippe Owezarski. Evaluating the impact of traffic sampling on aatac's ddos detection. In Proceedings of the 2018 Workshop on Traffic Measurements for Cybersecurity, pages 27–32, 2018.

[17] Karel Bartos, Martin Rehak, and Vojtech Krmicek. Optimizing flow sampling for network anomaly detection. In 2011 7th international wireless communications and mobile computing conference, pages 1304–1309. IEEE, 2011.

[18] João Marco C Silva, Paulo Carvalho, and Solange Rito Lima. A modular sampling framework for flexible traffic analysis. In 2015 23rd International Conference on Software, Telecommunications and Computer Networks (SoftCOM), pages 200–204. IEEE, 2015.

[19] Rayane El Sibai. Sampling, qualification and analysis of data streams. PhD thesis, Sorbonne Université; Université libanaise, 2018.

[20] Rayane El Sibai, Yousra Chabchoub, Jacques Demerjian, Raja Chiky, and Kablan Barbar. A performance evaluation of data streams sampling algorithms over a sliding window. In 2018 IEEE Middle East and North Africa Communications Conference (MENACOMM), pages 1–6. IEEE, 2018.

[21] Qiao Pan, Huang Yong-feng, and Zeng Pei-feng. Reduction of traffic sampling impact on anomaly detection. In 2012 7th International Conference on Computer Science & Education (ICCSE), pages 438–443. IEEE, 2012.

[22] Raman Singh, Harish Kumar, and RK Singla. Analyzing statistical effect of sampling on network traffic dataset. In ICT and Critical Infrastructure: Proceedings of the 48th Annual Convention of Computer Society of India-Vol I, pages 401–408. Springer, 2014.

[23] Rayane El Sibai, Yousra Chabchoub, Jacques Demerjian, Zakia Kazi-Aoul, and Kablan Barbar. Sampling algorithms in data stream environments. In 2016 International Conference on Digital Economy (ICDEc), pages 29–36. IEEE, 2016.

[24] William Cochran. *Sampling Techniques*. John Wiley & Sons, 1977.

[25] Pavlos S Efraimidis and Paul G Spirakis. *Weighted random sampling with a reservoir*. Information Processing Letters, pages 181–185, 2006.

[26] Pavlos S Efraimidis. *Weighted random sampling over data streams*. In Algorithms, Probability, Networks, and Games, pages 183–195. Springer, 2015.

[27] A Ian McLeod and David R Bellhouse. *A convenient algorithm for drawing a simple random sample*. Journal of the Royal Statistical Society. Series C (Applied Statistics), pages 182–184, 1983.

[28] Jeffrey S Vitter. *Random sampling with a reservoir*. ACM Transactions on Mathematical Software (TOMS), pages 37–57, 1985.

[29] Phillip B Gibbons, Yossi Matias, and Viswanath Poosala. *Fast incremental maintenance of approximate histograms*. In VLDB, volume 97, pages 466–475, 1997.

[30] Phillip B Gibbons, Yossi Matias, and Viswanath Poosala. *Fast incremental maintenance of approximate histograms*. ACM Transactions on Database Systems (TODS), pages 261–298, 2002.

[31] Brian Babcock, Mayur Datar, and Rajeev Motwani. *Sampling from a moving window over streaming data*. In Proceedings of the thirteenth annual ACM-SIAM symposium on Discrete algorithms, pages 633–634, 2002.

[32] Rayane El Sibai, Yousra Chabchoub, Jacques Demerjian, Zakia Kazi-Aoul, and Kabalan Barbar. *A performance study of the chain sampling algorithm*. In 2015 IEEE Seventh International Conference on Intelligent Computing and Information Systems (ICICIS), pages 487–494. IEEE, 2015.

[33] Rayane El Sibai, Jacques Bou Abdo, and Jacques Demerjian. *A new Priority Sampling Algorithm for the Internet of Things*. 2021.

[34] Rainer Gemulla, Wolfgang Lehner, and Peter J Haas. *A dip in the reservoir: Maintaining sample synopses of evolving datasets*. In Proceedings of the 32nd international conference on Very large data bases, pages 595–606. VLDB Endowment, 2006.

[35] Rainer Gemulla. *Sampling algorithms for evolving datasets*. PhD thesis, Technischen Universitat Dresden Fakultat Informatik, 2008.

[36] Baptiste Csernel, Fabrice Clerot, and Georges Hébrail. Datastream clustering over tilted windows through sampling. Knowledge discovery from data streams, page 127, 2006.

[37] Aboagela Dogman, Reza Saatchi, and Samir Al-Khayatt. An adaptive statistical sampling technique for computer network traffic. In 2010 7th International Symposium on Communication Systems, Networks & Digital Signal Processing (CSNDSP 2010), pages 479–483. IEEE, 2010.

[38] Aboagela Dogman and Reza Saatchi. Multimedia traffic quality of service management using statistical and artificial intelligence techniques. IET Circuits, Devices & Systems, 8(5):367–377, 2014.

[39] Mahbod Tavallaee, Ebrahim Bagheri, Wei Lu, and Ali A Ghorbani. A detailed analysis of the kdd cup 99 data set. In 2009 IEEE Symposium on Computational Intelligence for Security and Defense Applications, pages 1–6. IEEE, 2009.

[40] Sio-Iong Ao, Mahyar Amouzegar, and Burghard B Rieger. Intelligent Automation and Systems Engineering, volume 103. Springer Science & Business Media, 2011.

[41] Paul Hick, Emile Aben, Kc Claffy, and Josh Polterock. The caida ddos attack 2007 dataset, 2007.

[42] Markus Ring, Sarah Wunderlich, Dominik Grüdl, Dieter Landes, and Andreas Hotho. Flow-based benchmark data sets for intrusion detection. In Proceedings of the 16th European Conference on Cyber Warfare and Security. ACPI, pages 361–369, 2017.

[43] Jonathan Jedwab, Peter Phaal, and Bob Pinna. Traffic estimation for the largest sources on a network, using packet sampling with limited storage. Hewlett-Packard Laboratories, Technical Publications Department, 1992.

[44] Baek-Young Choi and Zhi-Li Zhang. Adaptive random sampling for traffic volume measurement. Telecommunication Systems, 34(1-2):71–80, 2007.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2021.3118605, IEEE Access

SUZAN HAJJ is a PhD student at University of Bourgogne - Franche-Comté (UBFC), France. She received her Master's Degree in Engineering from the Lebanese University in 2000. Her main research interests include Intrusion Detection Systems, Data Streams Pre-Processing, and Deep Learning.

RAYANE EL SIBAI received her master degree in software engineering from the Antonine University, Beirut, Lebanon, in 2014. She obtained her Ph.D. degree in computer sciences from the Pierre and Marie Curie - Sorbonne University, Paris, France, in 2018. Currently, she is an instructor at Al Maaref University, Beirut, Lebanon. Her research interests include Data streams processing, Data summarization, Anomaly detection, and Data quality.

JACQUES BOU ABDO is an interdisciplinary researcher with expertise in cybersecurity, blockchain, recommender systems, machine learning and network economics. Dr. Bou Abdo currently serves as assistant professor of cyber systems at University of Nebraska at Kearney. Previously, Dr. Bou Abdo served as assistant professor of computer science at Notre Dame University. He also served as Fulbright visiting scholar at the University of Kentucky. Dr. Bou Abdo is the founder of 2 technology startups specialized in cybersecurity and rural entrepreneurship. Dr. Bou Abdo holds a Ph.D. in Communication Engineering and Computer Science with emphasis on cybersecurity from Sorbonne University where he received the highest distinctions. He also holds a Ph.D. in Management Sciences from Paris-Saclay University.

JACQUES DEMERJIAN is a Full Professor of Computer Science and the director of LaRRIS (Laboratoire de Recherche en Réseaux, Informatique et Sécurité) research laboratory at the Faculty of Sciences at the Lebanese University (LU), Lebanon. He received his PhD degree in Network and Computer Science from TELECOM ParisTech-France in 2004. He has published more than seventy scientific articles in international journals / conferences / books chapters. His main research interests include Body Sensor Network, Intrusion Detection System and Mobile Cloud Computing. He is an IEEE Senior Member.

CHRISTOPHE GUYEUX obtained the agrégation in mathematics in 2001 and he defended his thesis in computer science at the University of Franche-Comté in 2010. He was recruited as an assistant professor in 2011, then as a Full Professor in 2014, same university. His work initially was about computer security and wireless sensor networks and is now focusing on artificial intelligence and bioinformatics. He has authored 90 international peer-reviewed journals and as many conference proceedings.

ABDALLAH MAKHOUL is a full professor in Computer Science at University of Bourgogne - Franche-Comté (UBFC), France. He received the PhD degree in computer science from the University of Franche-Comté (UFC), France, in 2008. From 2009 to 2019, he has been an Associate Professor with the University of Franche-Comté. He is a member of the DISC department (department of computer science and complex systems) of FEMTO-ST Institute, France. He is the head of the research team OMNI (Optimization, Mobility and NetworkIng). His research focuses upon the following areas: distributed algorithms, Internet of things, programmable matter, e-health monitoring and real-time issues in wireless sensor networks. He has been a TPC chair and member of several networking conferences and workshops and guest editor and reviewer for several international journals. He participated in several national and international research projects.

DOMINIQUE GINHAC received his Master's Degree in Engineering (1995) followed by a Ph.D in Computer Vision (1999) from Univ. Clermont Auvergne (France). He then joined Univ. Bourgogne as an assistant professor (2000) and was promoted to Full Professor of Computer Vision in 2009. He was head of the Le2i lab from 2016 to 2019. He has recognized expertise in embedded computer vision, computational imaging, and real-time image processing. Recently, he has become interested in deep learning on the edge applied to the analysis of human activities. He has authored 40 international peer-reviewed journals and over 100 conference proceedings.